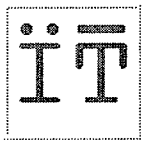
El Presidente,

Antonio J. de las Heras

UNIVERSIDAD CARLOS III DE MADRID



Departamento de Ingeniería Telemática

Doctorado en Tecnologías de las Comunicaciones

TESIS DOCTORAL

AUTOMATIZACIÓN DE TAREAS EN EL WEB: UNA PROPUESTA BASADA EN ESTÁNDARES

Autor: **Vicente Luque Centeno**
Licenciado en Informática

Directores: **Carlos Delgado Kloos** y **Luis Sánchez Fernández**
Doctores Ingenieros de Telecomunicación

CENTRO DE AMPLIACIÓN DE ESTUDIOS

D. VICENTE LUQUE CENTENO, con D. N. I. o pasaporte nº 11.825.418Z

AUTORIZA:

A que su tesis doctoral con el título: ***"Automatización de tareas en el web: una propuesta basada en estándares"*** pueda ser utilizada para fines de investigación por parte de la Universidad Carlos III de Madrid.

Leganés, 25 de junio de 2003

Fdo.: Vicente Luque Centeno

Tribunal nombrado por el Mgfco. y Excmo. Sr. Rector de la Universidad Carlos III de Madrid, el día 22 de Mayo de 2003

Presidente D. Antonio Rodríguez de las Heras

Vocal D. Antonio Hernández Pérez

Vocal D. Alberto Pan Bermúdez

Vocal D. José F. Aldana Montes

Secretario D. Andrés María López

Realizado el acto de defensa y lectura de la Tesis el día 22 de Mayo de 2003 en Leganés.

Calificación: Sobresaliente Cum Laude por Unanimidad

EL PRESIDENTE EL SECRETARIO LOS VOCALES

Índice general

Índice de cuadros

Índice de figuras

Agradecimientos

Quiero agradecer a todas aquellas personas que, de una un otra forma, han hecho posible este trabajo. Agradezco a mis padres el haber depositado su confianza en mí y haberme permitido seguir adelante y haberme apoyado en todo momento. A mis directores de tesis, quiero agradecerles los buenos y consejos y las sabias *preguntas* que me hicieron buscar las respuestas para elaborar este trabajo. A mis compañeros del Departamento de Ingeniería Telemática, por toda la colaboración recibida y por todo lo que de ellos he aprendido en estos años de doctorado. A mis antiguos compañeros de la Facultad de Informática, de donde tantísimas cosas aprendí. A mis amigos y amigas por haber estado ahí todo este tiempo.

Resumen

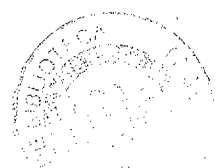
Actualmente, millones de usuarios en todo el mundo se ven abocados a realizar cada día tareas en Internet, manejando de forma repetida un conjunto cada vez mayor de remotas aplicaciones que se encuentran accesibles en el Web. Para todas esas labores, la gran mayoría de esas personas apenas cuenta con la única ayuda de los browsers, lo cual confiere al proceso de navegación una alta necesidad de interacción por parte del usuario, en lo que habitualmente se conoce como *navegación manual*. Sin embargo, ello a menudo implica un esfuerzo demasiado elevado para muchas personas, especialmente cuando el volumen de datos es grande o la complejidad de su manejo requiere muchas interacciones con el browser.

Desarrollar programas que naveguen automatizadamente por el Web, manipulando *inteligentemente* la información disponible en Internet es una necesidad cada vez más demandada en numerosos entornos. Sin embargo, el desarrollo de este tipo de programas tradicionalmente se ha afrontado con técnicas que implican un alto coste de desarrollo y un elevado coste de mantenimiento, además de tener una vida demasiado corta y ser muy sensibles a pequeñas modificaciones en las páginas accedidas. En este trabajo se proponen nuevas técnicas basadas en estándares para reducir el coste de estos desarrollos y mejorar la estabilidad de estos programas.

Abstract

Nowadays, millions of people around the world have to daily perform tasks on Internet, repeatedly managing an increasing amount of Web enabled applications. For many of these tasks, most people use browsers, by manually and mecanically interacting with screens of data retrieved from remote computers. However, this implies too much effort for too many people, specially when the amount of data is big or those data require complex managements with several user interactions.

Developing wrapper agents to automate these tasks for the user, by *intelligently* managing Web's data is an increasingly demanded need at several enterprises. However these programs have traditionally had a large development and maintenance cost, they have had very short lives and their behaviour when minor changes affect pages is not desirable. This document presents several standards-based new techniques for reducing these costs and making these programs more stable.



Capítulo 1

Planteamiento y objetivos

Esta tesis doctoral aborda el problema de la automatización de tareas en el Web. En cualquier automatización, como por ejemplo cualquiera de las llevadas a cabo en un complejo industrial, se persigue delegar en las máquinas la realización de trabajos, de forma que éstos sean llevados a cabo con la mínima intervención humana y mecánica posible bajo un coste amortizable.

El World Wide Web se ha convertido en poco tiempo en el mayor repositorio de conocimiento de la humanidad. Una multitud creciente de servicios Web, que comprenden, por una parte, a los servidores de información almacenada en simples páginas Web, y por otra, a las aplicaciones remotamente accesibles desde el Web para muy diversos propósitos, se puede acceder a través de una gran variedad de páginas Web. Gran cantidad de esa información se encuentra muchas veces almacenada en bases de datos que, siendo accesibles mediante interfaces específicamente diseñadas para el Web, se encuentran sin embargo a veces sin explorar conforme a las necesidades particulares de muchos usuarios. Mucha información, ciertamente, pero en especial muchas aplicaciones accesibles remotamente desde el Web, acaban quedando infrautilizadas para muchos usuarios en un mundo donde los browsers se han quedado ciertamente limitados como herramientas de trabajo, ya que carecen de cualquier tipo de propósito más allá que el de la mera presentación paginada de datos al usuario y la recogida de datos del usuario mediante formularios.

La heterogeneidad del Web actual es inmensa. Por un lado, hay innumerables formas posibles de organizar y estructurar los contenidos dentro de cada página mediante distintas combinaciones de etiquetas de marcado, no sólo HTML sino también XML. Por otro lado, existen múltiples variantes para distribuir la información entre las distintas páginas, así como múltiples

maneras de dejar éstas enlazadas entre sí. Toda esta heterogeneidad dificulta enormemente la integración de datos obtenidos de distintos servidores para su combinación o gestión unificada. La integración de datos de diversas fuentes, bien para la obtención de datos elaborados con valor añadido (sindicación de contenidos), bien como cadena de suministro (aplicaciones que deban pasarse datos de unas a otras), es un área que actualmente está demandando un creciente interés, habida cuenta de la necesidad de automatizar el manejo de grandes volúmenes de datos en el Web.

En ocasiones, los usuarios se encuentran con la tarea de tener que rellenar muchas veces un conjunto conocido de formularios o seguir de forma repetitiva un mismo conjunto de enlaces para poder conseguir la información que desean o para poder manejar intensivamente aplicaciones remotamente por medio del Web. Estos usuarios acaban invirtiendo para ello gran cantidad de tiempo en labores que, dado su automatismo, sería muchas veces posible que se realizaran autónonamente por un ordenador. Es habitual que muchas de estas aplicaciones del Web ofrezcan interfaces para realizar transacciones simples, pero no que dispongan de interfaces para ofrecer múltiples transacciones que puedan ser procesadas por lotes. Sin embargo, los usuarios no disponen hoy en día de otra herramienta más allá del browser, por lo que acaban debiendo aprobar con un click de ratón cada enlace que desean seguir una y otra vez, transacción a transacción, repetidamente cada vez que manejan una de estas aplicaciones accesibles desde el Web.

Tradicionalmente, uno de los grandes problemas del Web estriba en la incapacidad de manejar en él grandes volúmenes de información de forma efectiva, conforme a los deseos de los usuarios. Los grandes buscadores apenas resuelven una pequeña parte del problema de la búsqueda de información. Aunque son capaces de listar los documentos del Web en los que se sabe que aparecen los términos de búsqueda, no sin problemas como los apuntados por [28], los buscadores no dejan de devolver como resultados documentos enteros, dejando al usuario la responsabilidad de analizar su estructura, semántica o funcionalidad interna con el fin de buscar allí los datos que le interesan. Sin duda, las necesidades de los usuarios son mucho más complejas que aquellas a las que puede dar respuesta un simple buscador, ya que, una vez delante de la página en la que debe empezar a trabajar, debe ser el usuario quien indique por dónde navegar, construyendo así el camino que le lleve a conseguir sus objetivos.

1.1. Diferencias entre navegación manual y automática

Dentro del mundo industrial en general, la automatización persigue incrementar la productividad de las personas, minimizando los errores propios de la naturaleza humana y mejorando así la eficiencia en la realización de trabajos y en la optimización de los recursos involucrados. Con la automatización de tareas en el Web se persiguen esos mismos objetivos aplicados en la realización del cada vez mayor número de tareas que pueden realizarse a través del Web. Al igual que en cualquier empresa, el objetivo principal de la automatización de tareas en el Web es ahorrar tiempo y esfuerzo a las personas. En el caso del Web, los beneficios de la automatización serán más patentes en aquellas tareas que necesiten procesar grandes volúmenes de información o que deban ser frecuentemente ejecutadas.

Trasladando a las máquinas las actividades más automatizables y rutinarias, se permite ahorrar esfuerzo a las personas. Gracias a ello las personas pueden centrarse en otras actividades, habitualmente más productivas, creativas y probablemente más adecuadas para sus trabajos y sus preparaciones. Sin duda, la productividad y creatividad humana se ve muchas veces aletargada por las tareas más rutinarias de navegación en el Web. El hecho de tener que realizar una y otra vez los mismos pasos día a día delante de un browser, siguiendo los mismos enlaces y rellenando una y otra vez los mismos formularios, acaba siendo una tarea que requiere demasiado esfuerzo y dedicación para muchas personas. Ejemplos de tareas que pueden requerir ser realizadas frecuentemente, son las que se pueden llevar a cabo con aplicaciones como las bancarias (comprobaciones de saldo, transferencias, listado de movimientos, operaciones de bolsa, subastas de depósitos, fondos de inversión ...), de subastas (publicación de artículos para vender, búsqueda y comparación de artículos para pujar, inserción de pujas, gestión de avisos, evaluación de transacciones, ...), de compra-venta (búsqueda y comparación de artículos, realización de pedidos y de pagos, ...), de reserva de billetes de avión o de tren, habitaciones de hotel, entradas a espectáculos, envío de mensajes, así como un largo etcétera. Estas aplicaciones se encuentran cada vez más frecuentemente, tanto en las intranets de las empresas, como accesibles a todo el mundo en un gran número de servidores.

Algunas de las tareas Web, en especial las que deben hacerse de forma repetitiva mediante browsers, suponen demasiado esfuerzo, en tanto en cuanto para llevarlas a cabo, las acciones mecánicas que debe ejecutar el usuario para indicar sus instrucciones al ordenador, deben ser repetidas muchas veces.

Al contrario de lo que ocurre con la navegación manual basada en browsers, gracias a la automatización de tareas en el Web, grandes volúmenes de información distribuida en múltiples bases de datos accesibles desde el Web pueden ser procesados conforme a los intereses de los usuarios requiriendo de ellos un esfuerzo mínimo. Las principales diferencias entre la navegación manual y la automática, resumidas en la tabla 1.1, aparecen detalladas a continuación.

	Navegación manual	Navegación automática
Intervención	Humana	Ordenador
Acción	Mecánica	Programada
Esfuerzo	De navegación	De programación
Errores	De navegación	De programación
Tiempo de respuesta	Significativo	Ínfimo
Requisitos	Browser	Conexión
Repositorios	No programables	Programables
Procesamiento	Cálculo mental	Automatizable
Volumen de datos	Limitado	Enorme
Implantación	Factible	Costosa
Adaptabilidad ante cambios	Tolerable	Costosa

Cuadro 1.1: Diferencias entre la navegación manual y la navegación automática

1.1.1. Esfuerzo

El Web actual está diseñado para ser navegado de forma interactiva, con el usuario proporcionando mecánicamente una a una sus instrucciones detrás de la pantalla del ordenador, *haciendo click* en cada enlace que desea seguir. Ello supone en muchas ocasiones un serio coste de recursos humanos, esto es, de personas que deben dedicar a menudo una gran cantidad de tiempo, esfuerzo y perseverancia frente al ordenador para realizar tareas sencillas. Frente a esa opción, una tarea automatizada por un programa capaz de navegar en lugar del usuario, emulando el comportamiento de la actuación conjunta de éste y del navegador, puede reducir significativamente ese coste de navegación.

1.1.2. Propensión a errores

La interacción mecánica de las personas en la navegación manual aumenta en gran medida la propensión a cometer errores durante la ejecución de la tarea. Este riesgo se hace más probable cuando el conjunto de datos que debe ser manipulado es voluminoso. Por el contrario, un programa que navegue automáticamente por el Web puede manipular eficiente y adecuadamente grandes volúmenes de información, incluso a pesar de que ésta se encuentra distribuida en varias fuentes de datos.

1.1.3. Tiempo de respuesta

Pese a que el rendimiento de una aplicación Web está principalmente condicionado por el tiempo de respuesta del servidor y de las conexiones que comunican a éste con el cliente, lo cierto es que, las personas, cuando navegamos delante de un browser en el que debemos introducir interactivamente nuestras órdenes, tenemos unos tiempos de respuesta significativos. Además, y no menos importante, las personas somos fácilmente distraíbles de nuestros cometidos, como puede ocurrir con la aparición de enlaces que inesperadamente reclamen nuestra atención por un tema que nos interese y no tenga nada que ver con la tarea que nos estaba ocupando. Incluso en una escena en la que un operador se encuentre altamente concentrado en la realización de una tarea con un browser, el simple hecho de tener que activar mecánicamente unos dispositivos de introducción de datos, como el teclado y el ratón, estando a la vez pendiente de varias ventanas abiertas en la pantalla que reclaman simultáneamente la atención del usuario, supone unos tiempos de retraso que, aunque aceptables para unas pocas transacciones, resultan ciertamente frustrantes cuando el número de acciones mecánicas a ejecutar acaba siendo elevado, especialmente debido a la imposibilidad de ordenar trabajos por lotes en el Web.

1.1.4. Requisitos hardware y software

Sin duda, uno de los grandes problemas de los browsers es que, además de la intervención mecánica necesaria para poder proceder al seguimiento de enlaces, requieren de hardware y software especializado, de forma que gran parte del Web ahora mismo está pensada exclusivamente para ser accesible desde ordenadores personales (PC), con unas resoluciones de pantalla determinadas, y con unos requisitos de software específicos muy concretos.

Sin embargo, la proliferación de un cada vez mayor número de dispositivos electrónicos conectados a la red, como algunas cabinas públicas de acceso a Internet, los dispositivos inalámbricos, u otros pequeños electrodomésticos, está aumentando el número de dispositivos con reducidas capacidades de visualización de datos, que, sin embargo, no dejan de ser capaces de tratar adecuadamente los datos de la Red. Estos dispositivos podrían fácilmente conectarse a la red para automatizar tareas en el Web, al igual que lo hace cualquier otro ordenador, conforme a lo propuesto en [42], prescindiendo, al tratar automáticamente las páginas sin necesitar que el usuario las visualice para proporcionar interactivamente sus instrucciones sobre las mismas, de las capacidades de visualización que sí poseen los ordenadores de sobremesa en los que se ejecutan los browsers. Es decir, la navegación manual está prácticamente limitada a un conjunto muy particular de dispositivos (aquellos que tengan instalado el browser contemplado por el sitio Web), mientras que la navegación automática, al carecer de esas restricciones, puede llevarse a cabo desde un número mucho mayor de dispositivos de acceso, como neveras, asistentes personales, teléfonos móviles, set-top boxes ...

1.1.5. Adecuación de repositorios

Otro de los problemas con los que se encuentran las personas que navegan con browsers es que la forma de recolectar los datos relevantes que van encontrando está basada en métodos poco automatizables, e inadecuados para grandes volúmenes de datos, como la memorización, el apuntar los datos en un papel, guardar toda la página a fichero, imprimirla para analizarla frente a otras personas en una reunión, o, quizá en el mejor de los casos, el simple *copiar y pegar* en la ventana de otra aplicación. Sin embargo, ni el papel, ni la memoria humana, ni una ventana abierta de un editor de texto son repositorios adecuados para almacenar los datos relevantes que se van extrayendo de las páginas cuando el volumen de datos empieza a ser considerable. Por el contrario, desde el punto de vista de la automatización de tareas, es deseable la utilización de repositorios de datos accesibles desde programas de ordenador, como variables de memoria, ficheros con algún tipo de estructura conocida por el programador o registros en una base de datos.

1.1.6. Procesamiento

Pese a que muchas veces el procesamiento que se debe realizar sobre los datos del Web es realmente sencillo, (aunque ciertamente es posible que en

el futuro estos procesamientos se puedan volver cada vez más complejos), lo cierto es que el volumen de datos que muchas veces hay que manejar en el Web es demasiado alto como para ser manejado mentalmente por personas delante de un navegador. Es fácil encontrar la versión más barata de un libro en una página que tenga pocos ejemplares, pero ya no lo es tanto cuando se desean comparar un gran número de listados de tiendas, cada una con sus propios precios, de forma que haya que mezclar los resultados de todas ellas, y calcular el gasto final incluyendo gastos de envío, descuentos especiales o promocionales, o eliminando resultados repetidos. Realizar este tipo de tareas es algo que puede fácilmente ser automatizado por un programa capaz de acceder a los datos involucrados si éstos se encuentran convenientemente almacenados en un repositorio adecuado de datos como los mencionados en el punto anterior.

1.1.7. Coste de implantación y adaptabilidad

Todas las ventajas de la navegación automática frente a la navegación manual tienen un coste. Desarrollar aplicaciones que automaticen las tareas de navegación en el Web es costoso. Apenas existen técnicas específicamente orientadas a la reducción del coste de implantación de este tipo de programas. Buena parte de ellas puede encontrarse en el capítulo 3. Sin embargo, cualquier mínimo cambio en la estructura de las páginas involucradas en una tarea puede acabar requiriendo una ardua labor de mantenimiento. Por el contrario, las páginas cuya estructura es frecuentemente actualizada, no plantean excesivos problemas a las personas que las navegan manualmente con browsers, pues el comportamiento humano es fácilmente **adaptable** a las nuevas circunstancias, algo que no puede decirse normalmente del comportamiento de los programas de ordenador. En el apartado 1.5 aparecen más detallados estos costes.

1.2. Ejemplos de tareas costosas para la navegación manual

Según aumentan las posibilidades de realizar nuevas acciones en un mundo cada vez más interconectado, resulta cada vez menos difícil encontrar tareas capaces de absorber horas de trabajo a sus usuarios o en las que se justifique poco la conveniencia de que éstos deban estar presentes en todo momento ante la pantalla del ordenador. A continuación figura una lista de posibles

tareas Web que reflejan los problemas:

- Solicitar al Web de Hacienda el envío de datos fiscales de 100 personas. Actualmente esa labor se desarrolla rellenando un formulario en un único paso, pero que debe ser rellenado y enviado una vez para cada persona, introduciendo datos personales y fiscales de la declaración anterior.

- Publicar en eBay un catálogo de una tienda con 500 artículos en subastas. Actualmente cada artículo requiere el rellenado del orden de unos 8 ó 9 formularios, que aparecen en secuencia y que empiezan por la selección del tipo de subasta y categoría en la que se desea publicar, continúan preguntando por información especializada del artículo basada en las selecciones anteriores, y acaban solicitando una confirmación de que todo está conforme para el usuario. En algunos de los pasos cabe la posibilidad de que aparezcan variantes en la secuencia, como por ejemplo los que se desprenden del hecho de que cada artículo pueda incluir o no fotografía, por lo que para esos artículos puede ser necesario rellenar un formulario aparte para enviar la foto al servidor. Además, resulta deseable realizar una serie de mínimas comprobaciones en cada paso, como el cercioramiento de que las tarifas que se pretenden cobrar son las que el usuario espera, o que se confirme que cada artículo ha sido efectivamente puesto en subasta (no hacer este tipo de comprobaciones significa no tener garantizado el cumplimiento de la tarea). El algoritmo de esta tarea consistirá en el rellenado en secuencia de los formularios necesarios para la publicación de un artículo, contemplando las cuestiones mencionadas, de forma que esa secuencia figure dentro de un bucle que itere sobre los artículos que se desea publicar, presumiblemente leídos de algún repositorio definido por el usuario.

- Buscar con Google cada semana las páginas de cierto tema, manteniendo para ello una lista acumulada de páginas ya conocidas que se restará a los resultados obtenidos. De esta forma se presentará al usuario una lista que contenga exclusivamente las direcciones hasta el momento no conocidas por el usuario y que los robots de Google, en su continuo devenir por la red, van descubriendo. En todo caso, los resultados que se analicen de Google no deben comprender sólo la primera página de resultados, sino que debe inspeccionar todas independientemente de la paginación, filtrando de los resultados encontrados aquellos que aparezcan en la lista de conocidos por el usuario. Dicha lista deberá residir en un fichero en el ordenador del usuario, capaz de ser actualizado por

éste, pues Google, en su versión actual, no permite albergar este tipo de información personalizada para cada usuario.

- Componer un listado con los titulares de los principales periódicos cada mañana. Los resultados podrían ser visualizados en una televisión conectada a Internet mediante una set-top box que recoja los titulares de varias fuentes de noticias del Web y las presente agrupadas al usuario por secciones o según sus preferencias.

- Buscar, desde una nevera conectada a Internet, la mejor oferta de leche fresca en varias tiendas, incluyendo, a ser posible, no sólo la búsqueda en tiendas *habilitadas para neveras*, sino en cualquier tienda que venda a domicilio.

- Ampliar la tarea anterior para que busque la tienda que pide menos dinero por la entrega a domicilio de una cesta de la compra con varios artículos, incluyendo los gastos de envío y descuentos aplicables de cada tienda.

- Chequear si, entre los mensajes en varias cuentas de correo Web (de varios portales) hay algunos que cumplan ciertos requisitos, como remitentes especiales o fechas o asuntos determinados. A cada uno de esos mensajes, aplicarles ciertas acciones, como extraer de ellos los cuerpos de los mensajes para ser leídos, cambiarlos de carpeta, responderlos, reenviarlos a alguna dirección o simplemente borrarlos.

- Mandar un SMS (Short Message Service) a una lista de teléfonos móviles usando los formularios que para ello ponen accesibles algunos portales del Web, a ser posible eliminando las restricciones que imponen estos portales sobre tamaño de los mensajes y número de mensajes enviados por unidad de tiempo.

- Buscar en algún reconocido portal de ocio los restaurantes de una determinada zona geográfica, seleccionar aquellos que sirvan comida a domicilio y comparar sus ofertas.

- Combinar la información de restaurantes de un portal de ocio (que indique si aceptan tarjeta de crédito) con el callejero de la ciudad ofrecido por otro portal para obtener un listado de restaurantes que acepten tarjeta de crédito que estén cerca de la salida del cine al que voy a ir esta tarde.

La mayoría de los ejemplos anteriores está pensada para automatizar un elevado volumen de datos en una aplicación accesible desde el Web, como puede ser una aplicación de recolección de datos de la Administración, un buscador, una aplicación de publicación de subastas, una aplicación de envío de mensajes a móviles o una que gestione cuentas de correo electrónico desde el Web. En algunos casos, en lugar de acceder a una aplicación, la tarea debe acceder simplemente a varias páginas de estructura conocida pero de información variante, con el fin de combinar la información dispersa en esas fuentes de la forma deseada por el usuario.

1.3. Tipos de programas de navegación automatizada

Para conseguir automatizar tareas en el Web se necesita disponer de aplicaciones que automaticen la navegación de los usuarios en páginas y aplicaciones en el Web, de forma que sustituyan al usuario siguiendo alguna secuencia de enlaces que deban seguirse y formularios que deban rellenarse para poder realizar la tarea.

Los programas que pueden navegar en el Web pueden clasificarse según su grado de particularización a las páginas Web que se espera que visiten. Dicho criterio permite clasificar estos programas en tres grandes grupos:

1.3.1. Programas de navegación genérica no adaptada

Son programas que no están particularizados a ningún sitio Web y que, por lo tanto, pueden ser usados en cualquiera de ellos. Normalmente se dedican a solicitar interactivamente al usuario información acerca de qué enlaces seguir (browsers), o bien a seguir todos los enlaces de un sitio Web para realizar una tarea muy sencilla, normalmente idéntica en cada una de las páginas encontradas, como por ejemplo, la indexación por palabras de cada una de sus páginas (robots de buscadores), la comprobación de enlaces rotos [126], la descarga completa de sitios Web [20] o el aviso de que ciertas páginas han sido actualizadas recientemente [15]. Dado que estos programas carecen normalmente de un contexto semántico (ver tabla 1.6) y por lo tanto, son incapaces de particularizar su comportamiento al significado semántico de los datos de las páginas visitadas, sólo pueden ser usados en tareas muy sencillas, sin posibilidades de poder navegar de forma eficiente por el *deep Web* [110, 105].

1.3.2. Programas de navegación genérica adaptada

Son programas que, siendo en principio utilizables en cualquier Web, necesitan **meta-información** acerca del mismo para poder navegarlo de forma adaptada a sus características particulares. Normalmente, suelen analizar primero la meta-información, expresada en términos declarativos, del sitio Web para decidir *mientras navegan* qué enlaces tienen la *mayor* probabilidad de llevar a la consecución de unos objetivos preestablecidos. El Web Semántico [87] del W3C es un prometedor exponente de esta forma de automatización de tareas en el Web.

1.3.3. Programas de navegación particularizada

Son programas totalmente particularizados a las peculiaridades de un sitio Web concreto, por lo que, en principio, no son reutilizables en otros sitios Web. Estos programas, comúnmente denominados *wrappers* o código envoltorio [70], presentan importantes ventajas, ya que no necesitan ningún tipo de metadatos y pueden estar completamente adaptados a las características del sitio Web visitado. Mediante el seguimiento, particularizado al sitio Web, de enlaces pre-programados estáticamente según la semántica que el programador implícitamente refleja en el código de estos programas, prácticamente casi cualquier tarea concebida por el usuario puede ser desarrollada de forma eficiente, teniendo además en cuenta sus preferencias acerca de la forma en la que desea que se lleve a cabo la tarea. Por ejemplo, para borrar el correo *spam* en la bandeja de entrada de una cuenta de correo Web como el de Yahoo!, un usuario puede querer marcar los correos de ciertos dominios que él considera *spam*, para después pulsar en el botón de borrado, mientras que otro usuario puede preferir seleccionar los mensajes con determinados *asuntos* moviéndolos a una carpeta temporal que será borrada posteriormente. Sin embargo, este tipo de programas son muy costosos, tanto de desarrollar (debe implementarse uno para cada tarea), como de mantener (cualquier cambio en la estructura de las páginas accedidas puede provocar un malfuncionamiento del programa).

En la tabla 1.2 aparecen resumidas las principales características de los distintos tipos de programas que navegan por el Web según su grado de adaptación a las páginas de esos sitios Web.

	Genérica no adaptada	Genérica adaptada	Particularizada
Ejemplos	Browsers, robots	Web Semántico	Wrappers
Algoritmo	Usuario, fuerza bruta	Guiado por objetivos	Pre-programado
Contexto semántico	Nulo	Metadatos declarativos	Implícito en programación
Construcción de caminos	Usuario, todos	Dinámica	Estática, pero flexible
Implantabilidad	Fácil	Incipiente, a largo plazo	Conocida y factible
Mantenimiento	Ínfimo	Fácil (declarativo)	Costoso (programación)
Aplicable a cualquier Web	Sí	Si tiene metadatos	No
Automatización de tareas	MUY simples	No complejas	Sí

Cuadro 1.2: Clasificación de programas que navegan por el Web según su adaptación

Comparaciones de costes

Siempre que se permita al autor un par de pequeñas licencias literarias, cabría quizá comparar los distintos tipos de programas orientados a la navegación en el Web con soluciones médicas que intentan paliar una enfermedad en una persona y con la navegación marina.

Los programas genéricos no adaptados podrían asimilarse a los tratamientos médicos más sencillos, sin receta y con un bajo coste, y que todo el mundo puede usar, como el hecho de beber agua, ingerir té verde o incluso tomar una aspirina. Son éstas soluciones válidas y fácilmente aplicables para problemas habitualmente comunes de encontrar. Los programas genéricos adaptados podrían quizá asimilarse a una fisioterapia o a un tratamiento medicinal con receta médica. Este tipo de soluciones está enfocada a una finalidad más específica que los tratamientos sencillos son incapaces de resolver por sí mismos, por lo que se requiere normalmente la ayuda de alguien externo capaz de facilitar la solución al usuario. Finalmente, los programas no genéricos, esto es, los completamente particularizados, pueden ser comparables a una operación de cirugía. Este tipo de operaciones tiene un coste sensiblemente superior al de otras alternativas, pero es realmente efectivo cuando el problema del usuario no puede ser solucionado con las técnicas anteriores.

De la misma forma, puede hacerse un símil de la navegación en el *deep Web* con los viajes en el mar. Una navegación basada en browsers puede ser asimilada con un buzo, capaz de sumergirse con autonomía propia por cualquier recoveco marino por estrecho que sea, pero incapaz de recorrer grandes distancias ni de llegar a grandes profundidades (además de estar frecuentemente orientada al ocio). Los robots son asimilables a los barcos, capaces de

recorrer grandes distancias en superficie, pero incapaces de navegar por el *deep Web* rellenando formularios o siguiendo enlaces de profundidad potencialmente ilimitada. La gran diversidad de robots según su complejidad es asimilable a la gran diversidad de barcos existentes, desde pequeñas balsas a transatlánticos, según su complejidad. En todos los casos, su navegación está centrada en la parte más superficial del mar y nunca se sumergen en el fondo del *deep Web*. La navegación con alternativas genéricas adaptadas como la del Web Semántico es asimilable a varias alternativas, según el punto de vista de sus promotores o sus detractores. Para los primeros, el Web Semántico es comparable a un enorme submarino tripulado, capaz de automatizar cualquier tarea a cualquier distancia y profundidad y con capacidad autónoma (programación de agentes y de inteligencia artificial) para tomar decisiones durante la navegación, pero incapaz de introducirse por los recovecos en los que no cabe, es decir, incapaz de navegar sin sus correspondientes metadatos. Para sus detractores, sin embargo, el Web Semántico promete demasiadas cosas irrealizables aún hoy en día, y es en realidad más asimilable al uso de una red de transitables túneles construidos debajo del fondo del mar (como el Eurotunel que atraviesa el canal de la Mancha). En ambos casos se requiere la construcción de rutas de navegación (por las que quepa el submarino) o transitables túneles adaptados al fondo marino, que sean asimilables a los metadatos adaptados al sitio Web al que describen. Estos metadatos actúan como guías de la navegación, indicando por dónde navegar y por dónde no y cuál es el camino que hay que tomar en cada bifurcación para llegar a un destino. Finalmente, cuando se necesita recorrer grandes distancias, acceder a grandes profundidades e introducirse por pequeños recovecos en los que no cabe un enorme submarino, y no se dispone de metadatos, lo habitual es recurrir a soluciones como los pequeños batiscafos no tripulados y teledirigidos, costosos, pero capaces de recorrer cualquier ruta. La tabla 1.3 contiene un resumen de estas comparaciones aclarativas.

	Genérica no adaptada	**Genérica adaptada**	**Particularizada**
Ejemplos	Browsers, robots	Web Semántico	Wrappers
Medicina	Té verde, aspirinas, ...	Medicación con receta, fisioterapia, ...	Cirugía
Deep Web	Buzo, barcos	Red de túneles bajo el mar, submarino	Batiscafo no tripulado
Coste	Bajo	Medio/alto	Alto

Cuadro 1.3: Comparaciones aclarativas de alternativas de navegación según coste

Las tareas que los usuarios necesitan automatizar en el Web pueden llegar a ser bastante complejas. Sólo aquellos programas capaces de manipular adecuadamente los datos que aparecen en los documentos visitados pueden servir al usuario para automatizar sus tareas en el Web. Dado que la mayoría de las tareas realizables por las aplicaciones de navegación genérica no adaptada son demasiado simples y que el Web Semántico es aún una tecnología realmente incipiente, la principal alternativa utilizable actualmente consiste en la creación de programas de navegación particularizada a los sitios Web.

La navegación basada en wrappers ha sido sin duda la que más aportaciones ha realizado últimamente a la automatización de complejas tareas en el Web, en donde ha demostrado en repetidas ocasiones ser aplicable de forma fructífera [67]. Si bien los contextos semánticos (ver tabla 1.6) basados en metadatos declarativos pueden aún tardar tiempo en ser efectivos, es constatable que los contextos semánticos embebidos en las sentencias de programación que forman parte del código de un wrapper son una poderosa arma de automatización. Por otro lado, si bien la construcción dinámica de caminos de navegación guiados por el cumplimiento de objetivos resulta impracticable sin la existencia de metadatos, la construcción estática, pero flexible para ganar robustez, de caminos pre-programados de navegación donde no sea necesaria la existencia de metadatos que indiquen los enlaces que se deben seguir, puede servir como base para la automatización de tareas que manejen aplicaciones accesibles desde el Web.

1.3.4. Modos de integración de aplicaciones Web

Cuando se desea automatizar tareas que involucran a más de una aplicación Web, dos modos de integrarlas suelen ser los más empleados.

Integración de aplicaciones similares

Es normal encontrar aplicaciones de diversos servidores que, siendo semánticamente similares porque están concebidos para fines parecidos, desde el punto de vista del marcado de sus páginas, son estructuralmente distintas. Normalmente, se trata de aplicaciones que, aunque pueden tener pequeñas diferencias entre sí, ofrecen una funcionalidad similar al usuario, cada una accesible desde su propio servidor. Es por ello que el usuario puede tener que conectarse simultáneamente a varias de ellas a la vez, escogiendo *sobre la marcha*, qué acciones desea ejecutar en ciertas aplicaciones y qué otras acciones desea ejecutar en otras. Agentes de bolsa, personas con varias cuentas

de correo electrónico o con varias cuentas bancarias, personas que manejen varios sitios de subastas en el Web o que quieran simplemente buscar las mejores ofertas en varias tiendas suelen tener que abrir varias ventanas, una para cada una de estas aplicaciones, y operar con todas ellas a la vez.

Integración de aplicaciones complementarias

Por otro lado, también resulta frecuente encontrarse con tareas que involucran el intercambio de datos entre varias aplicaciones construidas de forma independiente unas de otras, pero que no estén integradas convenientemente, por lo que la única forma de manejarlas sea manipulando los formularios propios de acceso a la herramienta. Ante esta situación, los usuarios típicamente deben conectarse a alguna de ellas y, tras analizar los datos obtenidos de ésta, enviar esos mismos datos, quizá con alguna variación, a otra herramienta, rellenando con el teclado y el ratón su correspondiente formulario. Eso suele ocurrir en entornos donde, por ejemplo, una aplicación se dedica a recoger peticiones de muchos usuarios y, antes de procesarlas convenientemente, un supervisor debe darles el visto bueno, reinsertando esos datos en otra aplicación junto con alguna información adicional. En ese caso, el supervisor, si las herramientas no se han integrado convenientemente, puede tener que estar copiando los datos de una ventana de la primera herramienta y pegándolos en una ventana de la segunda herramienta. Otro ejemplo distinto, pero con el mismo planteamiento operativo, se puede encontrar en personas que deban enviar por alguna aplicación Web de correo electrónico determinados fragmentos de documentos que una primera aplicación les pueda estar mostrando en otra ventana. Sin duda, las tareas realizables pueden llegar a ser ciertamente impensables para los diseñadores de las aplicaciones Web accedidas, pues son los usuarios quienes mejor definen la forma en la que desean hacer sus tareas. Por ejemplo, una persona puede desear responder desde el correo que tiene en Yahoo! los correos electrónicos que le lleguen a su cuenta de correo en Hotmail, porque no desee responder directamente desde allí. En este caso, los formularios de Yahoo! deberán recibir información, no sólo del usuario, sino de la proporcionada por Hotmail, de la misma forma en la que un usuario que realizara esa tarea hubiera *copiado y pegado* ese texto desde la ventana de una aplicación a la ventana de la otra.

1.3.5. Sistemas mediadores

Los datos semiestructurados, que aparecen detallados en el apartado 1.4.7, aunque primordialmente se encuentran en la inmensa multitud de páginas HTML disponibles dentro de todo el Web, también pueden encontrarse con frecuencia en abundantes fuentes de datos dentro de entornos corporativos de tamaño mediano o grande. Precisamente por ello, son numerosas las empresas que presentan actualmente necesidades de poder integrar, no ya en un único documento, sino quizá también en la toma de decisiones operativas de la empresa, los datos provenientes de varias de esas fuentes de datos preexistentes, de la misma forma en la que es necesario realizar tareas que combinen la información obtenida de distintas fuentes en el Web.

Si bien es habitual que cada una de esas fuentes fuera desarrollada en su momento con distintas tecnologías y objetivos, lo cierto es que la gran mayoría de ellas fue concebida para ser accedida de forma que la información resultante de cada una de ellas fuera a ser consultada directamente, bien por personas, bien, en el mejor de los casos, pese a su enorme coste, por otras aplicaciones capaces de realizar llamadas a una interfaz particular de la aplicación. Las nuevas formas de negocio de hoy en día y la necesidad de manipular mucho más eficientemente grandes volúmenes de información de fuentes muy diversas han contribuido ineludiblemente a una necesidad cada vez mayor de que esas consultas a los antiguos sistemas aislados preexistentes puedan estar ahora integradas en otras consultas realizadas por nuevos programas capaces de combinar eficientemente la información que pueda estar distribuida en varios repositorios independientes entre sí.

De esta forma, los datos relevantes se pueden considerar como provenientes de fuentes semi-estructuradas [66], esto es, con un formato reconocible más allá del simple texto sin estructura, si se tienen en cuenta precisamente combinaciones especiales de etiquetas de marcado que son capaces de identificar a los datos clave que deben extraerse de las páginas.

Un foco importante de investigación en la actualidad es la construcción de sistemas capaces de integrar de manera sencilla datos semiestructurados heterogéneos y dispersos de múltiples fuentes accesibles por medios telemáticos de forma que se obtenga de los mismos una visión similar a la proporcionada por una única base de datos convencional. La idea de este enfoque consiste en ocultar todo tipo de heterogeneidad de cada una de las fuentes de datos accedidas de forma que al usuario se le proporcione una única visión global de todo el sistema y pueda manipular cada uno de los recursos de la red de una forma uniformizada, pese a que cada sistema trabaje en realidad de

forma distinta y tenga sus propias particularidades sintácticas. Los sistemas Datawarehouse, basados en la replicación de información a un repositorio común [68, 92] usados en entornos cerrados desde hace tiempo, no son adecuadamente escalables para ser usados en el Web. Quizá la aproximación arquitectural más utilizada hasta el momento para este tipo de sistemas sea la de los sistemas mediadores [79, 101]. En ella, los datos permanecen en sus fuentes originales y un sistema intermedio, llamado mediador, se encarga de proporcionar a lo usuarios la ilusión de que existe una única fuente en la que se encuentran todos los datos combinados y unificados de manera coherente de acuerdo a un único esquema global. Cuando el mediador recibe una consulta sobre el esquema global, ésta se reformula en diversas subconsultas que se realizan directamente sobre las fuentes originales. La interacción directa con las fuentes es delegada por el mediador a los llamados programas envoltorio (o *wrappers*), cada uno de ellos especializado en el diálogo concreto con cada servidor. Estos se encargan de recibir las peticiones del mediador, traducirlas como subconsultas ajustadas al formato particular de cada fuente, ejecutarlas sobre la misma y obtener los resultados, devolviéndoselos al mediador. Es entonces cuando los resultados de las fuentes son reestructurados por el mediador para ajustarse al esquema global y ser devueltos entonces al usuario. De esta manera, éste obtiene la impresión de estar consultando un único sistema. Este esquema se encuentra representado en la figura 1.1.

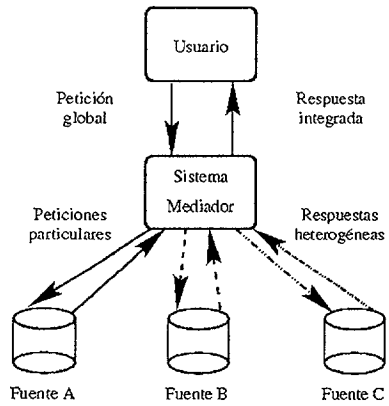


Figura 1.1: Sistema mediador

La motivación de la construcción de este tipo de sistemas se encuentra tanto en los planes estratégicos a nivel corporativo dentro de una organización, como a nivel de servicios utilizables por personas de forma individual. Las organizaciones se encuentran, por un lado, con que su operativa genera

una cantidad de información enorme, que es difícil de gestionar, y a la que es complicado, cuando no imposible, sacar el máximo partido. Por otro lado, diversos factores, como la rápida evolución tecnológica, las presiones del mercado o el efecto de la competencia entre diferentes proveedores de soluciones técnicas, han causado que los entornos y las tecnologías utilizadas para la generación y manejo de estos datos difiera en gran medida a lo largo del tiempo, creando así, a lo largo de los últimos años, un escenario donde grandes volúmenes de información se encuentra dispersa en fuentes independientes y heterogéneas, cuyos datos necesitan ahora más que nunca ser combinados y unificados. Son muchas las organizaciones que están desarrollando o tienen planes de desarrollar sistemas capaces de proporcionar visiones unificadas de los datos manejados por la organización.

Los problemas a los que los sistemas mediadores se han venido enfrentando en los últimos años son extrapolables a la realización de muchas tareas en el Web. Muchos de las aplicaciones construidas en las intranets de los sistemas corporativos a lo largo de los últimos años han sido creadas con interfaces para el Web. Es por ello que los sistemas mediadores se han visto últimamente cada vez más abocados a la automatización de tareas en las aplicaciones de las intranets de estas corporaciones. A pesar de que el número de posibles tareas automatizables en el Web es mayor que las realizables en las intranets de las aplicaciones corporativas, los sistemas mediadores, impulsados por las necesidades de estas corporaciones, no han dejado de ser un importante exponente de la necesidad de automatizar el tratamiento de datos obtenible de aplicaciones accesibles desde el Web.

1.3.6. Asistentes de navegación Web

Para automatizar tareas en el Web se necesitan programas capaces de eliminar la necesidad de que el usuario deba interactuar incansablemente con el ordenador durante la ejecución de la tarea. En lugar de solicitar que, durante la navegación, el usuario active el seguimiento de cada enlace que se desea visitar, se deben preprogramar todas esas activaciones en un algoritmo, de forma que queden así explícitamente la selección *preaprobada* de los enlaces que se seguirán durante la ejecución de la tarea. El resultado de ese algoritmo normalmente consistirá en presentarle al usuario sólo los datos (resultados, confirmaciones, ...) que a éste le interesan, evitando, en lo posible, mostrarle toda aquella información irrelevante para su tarea. El programa capaz de automatizar de esta forma las tareas del usuario conforme a sus intereses se denomina **asistente de navegación Web**. En la figura 1.2 aparece es-

quematizada la diferencia entre la tradicional navegación manual basada en browsers y la navegación basada en asistentes de navegación Web. Con un asistente de navegación Web, se persigue sustituir al browser y al usuario que lo maneja por un único programa capaz de automatizar una tarea de forma que los servidores Web involucrados presten a la aplicación el mismo servicio que prestarían a un usuario que realizara esa tarea con un browser.

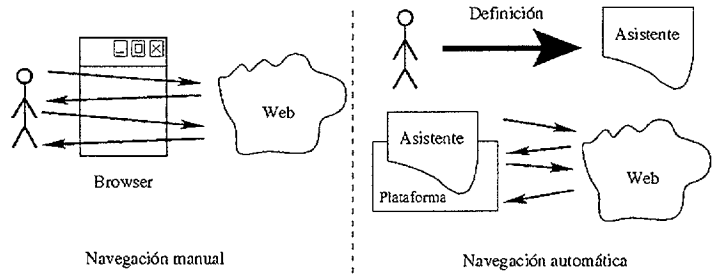


Figura 1.2: Comparación de navegación manual con automática

1.4. Características de los datos del Web

Para procesar datos automatizadamente hay que tener en cuenta las principales características de los datos cuyo tratamiento se pretende automatizar. A continuación se presentan algunas de las principales características de los datos del Web.

1.4.1. Voluminosidad

En las últimas décadas las nuevas tecnologías han permitido la creación, recopilación y publicación de grandes volúmenes de información accesible en una amplia diversidad de formatos y mediante una amplia gama de medios de acceso telemáticos. Por otra parte, muchas aplicaciones ya existentes en las intranets de muchas organizaciones han sido reconvertidas para poder ser accedidas ahora desde el Web, mediante un mayor número de terminales. Esto ha llevado, por otra parte, a la aparición de un gran número de aplicaciones que, siendo accesibles desde cualquier punto del planeta mediante el Web, a través de sus interfaces basadas en formularios, proporcionan al Web una enorme cantidad de información para la que no existen apenas medios adecuados de manipulación.

1.4.2. Heterogeneidad

El Web es un ente tremendamente heterogéneo, lo cual ha implicado una gran dificultad a la hora de manejar adecuadamente los grandes volúmenes de información dispersa en el Web. La proliferación de fuentes de información desarrolladas de forma totalmente independiente unas respecto de las otras ha llevado a un escenario en el que mucha información potencialmente útil para las necesidades particulares de muchos usuarios esté dispersa y almacenada según esquemas y estructuras de almacenamiento con grandes heterogeneidades. Debido a la intrínseca naturaleza abierta en el Web, los contenidos de la misma se muestran débilmente estructurados y, cuando lo están, su esquema es enormemente heterogéneo, presenta irregularidades, y sigue muy pocas restricciones. Otra peculiaridad del Web actual es el hecho de que la inmensa mayoría de su información es legible para las personas, pero no lo es para las máquinas, por lo que su tratamiento automatizado se encuentra con importantes problemas. La heterogeneidad de las Webs de las empresas supone aún mayores problemas cuando se las pretende interconectar con las de sus clientes o proveedores u otros canales de comercialización.

1.4.3. Orientación a los visualización

Una de las grandes barreras a la automatización en el Web actual (no tanto por los principios en los que se fundó, sino por el uso que se le ha ido dando después) es que está muy orientado a la visualización. La gran mayoría de los diseñadores no han concebido sus páginas para ser procesadas por otro tipo de aplicaciones distintas a los browsers. Es más, tampoco resulta extraño que algunos diseñadores hayan concebido sus páginas para que sean navegadas exclusivamente desde **ciertos browsers concretos**, excluyendo, en ocasiones sin saberlo, en otras intencionadamente, la navegación de otros tipos de programas como las que aparecen en la tabla 1.2.

La actual orientación a la visualización en el Web no es un problema en sí mismo para la accesibilidad si esa orientación está basada en hojas de estilo. Sin embargo, quizá porque las hojas de estilo nacieron algo más tarde que el propio Web y aún no tienen un soporte completo en muchas herramientas, tradicionalmente las páginas han reflejado su orientación a la visualización en el propio marcado HTML de sus páginas. La gran mayoría de los sitios Web actuales centran aún el uso de su marcado HTML en aspectos primordialmente visuales (tamaños y tipos de letra, colores, alineamientos, espaciados, distribución por la pantalla, imágenes, ...), muchos de ellos enfocados para impactar visualmente al usuario que use uno de los browsers

gráficos admitidos por el diseñador del sitio Web.

A pesar de que iniciativas como las de las hojas de estilo CSS bien permiten independizar a HTML de esa orientación a la visualización, buena parte del marcado HTML que se usa en las páginas de millones de aplicaciones accesibles desde el Web, creadas a lo largo de los últimos años, está aún orientado a estos fines. Así, en lugar de un Web en la que cualquier página HTML pueda ser navegable desde cualquier browser y dispositivo de acceso, donde las distintas peculiaridades de cada uno estén contempladas en distintas hojas de estilo, cada una de ellas adaptada a un tipo distinto de terminal, el Web actual adolece de un grave problema de orientación a un reducido número de browsers, puesto que los usuarios que utilizan otros tipos de programas de acceso distinto a esos browsers, no tienen un adecuado acceso a los contenidos que desean visitar. Para muchos usuarios, son muchas las páginas inaccesibles, donde determinados contenidos dejan de ser visibles y muchas aplicaciones accesibles desde el Web dejan de ser funcionales simplemente por el hecho de que se acceda a ellas con browsers no considerados por el diseñador del sitio Web.

Por ejemplo, resulta habitual el hecho de que se visualicen enlaces que sin embargo no se pueden pulsar (tienen una capa invisible encima que lo impide), textos que quedan ilegibles por aparecer superpuestos unos sobre otros, o incluso servidores que, siendo en realidad visitables, discriminan según el agente de usuario empleado y acaban redirigiendo a la portada del Web o a una página de error en la correspondiente respuesta a toda petición en la que no se acompañe la identificación del browser esperado en ese Web. En muchos sitios, la navegación basada en browsers sobre terminales tipo texto, como Lynx [10] resulta impracticable. Incluso la navegación basada en browsers gráficos resulta impracticable en muchos sitios Web si no se tiene activada alguna funcionalidad especial que el diseñador del sitio Web consideró como necesaria, como son las cookies, las rutinas JavaScript [74], las imágenes, o las animaciones en Flash u otros formatos. A ello hay que añadir la escasez práctica de descriptores textuales adecuados en numerosos componentes de las páginas, como los scripts, los applets, los plugins, los frames o los controles ActiveX, muchos de los cuales sólo resultan funcionales en determinados browsers, pero no en otras aplicaciones.

Accesibilidad

Iniciativas de mejora para la accesibilidad, como las publicadas por el W3C en sus recomendaciones [128, 137, 136], están encontrando últimamen-

te un decidido apoyo en numerosas personas e instituciones para conseguir que el acceso a los contenidos Web sea funcional con independencia de cual sea la plataforma de acceso, tanto hardware como software. Estas recomendaciones últimamente hacen especial hincapié en poco costosas, pero efectivas mejoras de acceso al Web por parte de, tanto browsers específicos usados por personas discapacitadas, como los recientes terminales inalámbricos con pantallas de reducidas dimensiones y escasa capacidad de procesamiento o ancho de banda limitado, tales como teléfonos móviles, asistentes personales o pequeños electrodomésticos. La correcta operatividad con independencia del browser utilizado o del dispositivo de acceso escogido es una de las grandes necesidades del Web actual. Últimamente, el cada vez mayor número de formas de acceso ha puesto aún más de manifiesto el serio problema de accesibilidad existente en el Web, como lo demuestra el hecho de que los nuevos dispositivos sean incapaces de acceder adecuadamente a las páginas del Web *legado*. Este problema ha permanecido *disimulado* por el importante dominio de Microsoft Internet Explorer como browser más utilizado en los últimos años. No obstante, numerosas iniciativas de mejora, como las del W3C y otras [109, 106, 96, 121] han sido propuestas para aportar soluciones. Dentro de las recomendaciones del W3C se enumeran varios tipos de programas de navegación, mucho más allá de los habituales browsers gráficos. El objetivo de estas recomendaciones es la de permitir el correcto acceso al Web desde terminales en modo texto, software específico de navegación para personas con visión discapacitada que presentan al usuario las páginas en Braille o en audio, así como desde terminales de reducidas dimensiones, con pequeños displays como los cada vez más habituales dispositivos inalámbricos, y muchos más.

El objetivo principal de estas iniciativas consiste en que las páginas publicadas en el Web sean accesibles a sus usuarios independientemente del dispositivo de acceso empleado por cada uno de ellos. Una de las ideas principales de estas iniciativas de mejora consiste en reducir al mínimo el uso del marcado HTML que esté orientado a la visualización, delegando esa tarea en las hojas de estilo y simplificando así el marcado estructural de las páginas. En la tabla 1.4 pueden contemplarse las principales diferencias entre las distintas versiones de HTML surgidas en los últimos tiempos, desde los puntos de vista de la orientación a la visualización y a la accesibilidad.

Resulta curioso destacar en este punto cómo los sitios Web que más éxito han cosechado en los últimos tiempos (Google, Yahoo, eBay, EasyJet ...) suelen tener como característica común el hecho de que no hacen apenas

HE = Hojas de estilo	HTML sin HE	HTML con HE	XHTML o XML
Orientación a la visualización	En HTML	En hoja de estilo	En hoja de estilo
Accesibilidad por terminales	Baja	Media	Alta
Regularidad estructural	Orientación visual	Sencilla	Sencilla
Modificaciones en la estructura	Habituales	Escasas	Ínfimas
Reglas de construcción	Básicas	Básicas	Tipo de documento
Automatización de tareas	Difícil	Media	Fácil
Difusión relativa en el Web	Alta	Baja	Muy baja

Cuadro 1.4: Principales diferencias entre las últimas versiones de HTML

uso de tecnologías que mermen la accesibilidad de sus páginas. Sin embargo, la escasa implantación práctica de las normas y recomendaciones del W3C durante muchos años hace prever que la transición hacia un Web accesible para browsers de distintos terminales será aún lenta, en cuanto a la inercia del Web está asentada en su gran tamaño y en la tradicional escasa orientación de diseñadores y herramientas a estos aspectos. La orientación del Web a los browsers como únicas herramientas de acceso es la razón de que el Web sea aún muy abundante en páginas que no sólo no cumplen las reglas de la recomendación oficial de XHTML, sino que ni tan siquiera cumplen muchas de las reglas de las recomendaciones anteriores de HTML.

1.4.4. Relevancia dependiente de la tarea

Por datos relevantes se debe entender, no sólo aquella información específica que está siendo buscada y que forma parte de los resultados que, quizá tras algún procesamiento, se le deben proporcionar al usuario al finalizar la tarea, sino, en general, todos aquellos datos que puedan ser utilizables en algún momento dentro de ese proceso de navegación. Ejemplos de datos relevantes son las direcciones de las páginas que se deben visitar, los campos de formularios que se deben rellenar, o quizá incluso simples campos que aparecen en las páginas en función de cuyos valores se puede tomar la decisión de seguir uno u otro enlace o efectuar una u otra acción definible por el usuario. No toda la información de cada página Web es igualmente relevante. Es común que, para una tarea particular, de una sola página apenas interese un dato, un determinado enlace o un formulario concreto y los demás datos de la página puedan ser felizmente ignorados. La relevancia de cada información, por otra parte, no se puede considerar en términos absolutos. La relevancia de un dato depende de la tarea concreta que se desee realizar, es decir, de los objetivos por los cuales la página que contiene el dato ha sido consultada.

Por ejemplo, dentro de un mismo documento, para una tarea concreta, puede ser interesante un enlace, mientras que para otra tarea, puede serlo otro enlace completamente distinto. Para ciertas personas, determinada información puede ser considerada como relevante, mientras que, para otras, pese a que tengan propósitos similares, esa misma información puede no serlo. En cualquier caso, dicha información, relevante o no, es presentada al usuario típicamente embebida en páginas Web que, estando escritas en HTML para su visualización en browsers, no están, sin embargo, orientadas para ser procesadas por otro tipo de aplicaciones capaces de manipular esa información automáticamente. Es decir, las páginas HTML están construidas para ser visualizadas, pero no para ser *comprendidas* por máquinas. El problema de la integración de datos del Web en aplicaciones, esto es, de su automatización, ha sido abordado de manera muy activa por investigadores de diferentes comunidades y disciplinas, tales como la Telemática, la Minería de Datos o la Inteligencia Artificial, realizándose en los últimos años importantes contribuciones desde diferentes puntos de vista.

1.4.5. Regularidad estructural

Las bases de datos, que almacenan su información de forma estructurada, al volcar sus contenidos en páginas Web, suelen conservar en ese volcado cierta regularidad, denominada *estructural*, en el marcado HTML de los datos volcados. Distintos datos conviviendo bajo la misma férrea estructura de una base de datos acaban siendo volcados al Web con similares etiquetas HTML, por lo que conservan una cierta regularidad estructural en sus marcados. Un ejemplo de ello puede verse en la figura 1.3, donde se muestra parte del código de una página Web que contiene los resultados de una búsqueda en Google.

El hecho de que exista una regularidad estructural en las páginas HTML publicadas en un mismo servicio (aplicación accesible desde el Web o conjunto de páginas lógicamente enlazadas para un fin común y publicadas bajo una misma estructura) tiene un inmenso valor para la extracción de datos del Web. La regularidad estructural puede ser usada para identificar unos datos respecto de otros. Sin embargo, el hecho de que dicha estructura no sea explícita (como sí lo es el esquema de una base de datos), sino implícita, permite que los sitios Web puedan cambiar en cualquier momento sus estructuras de marcado HTML sin apenas contemplaciones. Además, la estructura supuestamente regular de los sitios Web puede presentar irregularidades con frecuencia. Por ejemplo, es habitual que en el catálogo electrónico de una tienda todos los artículos contengan datos como precios, descripciones o marcas,

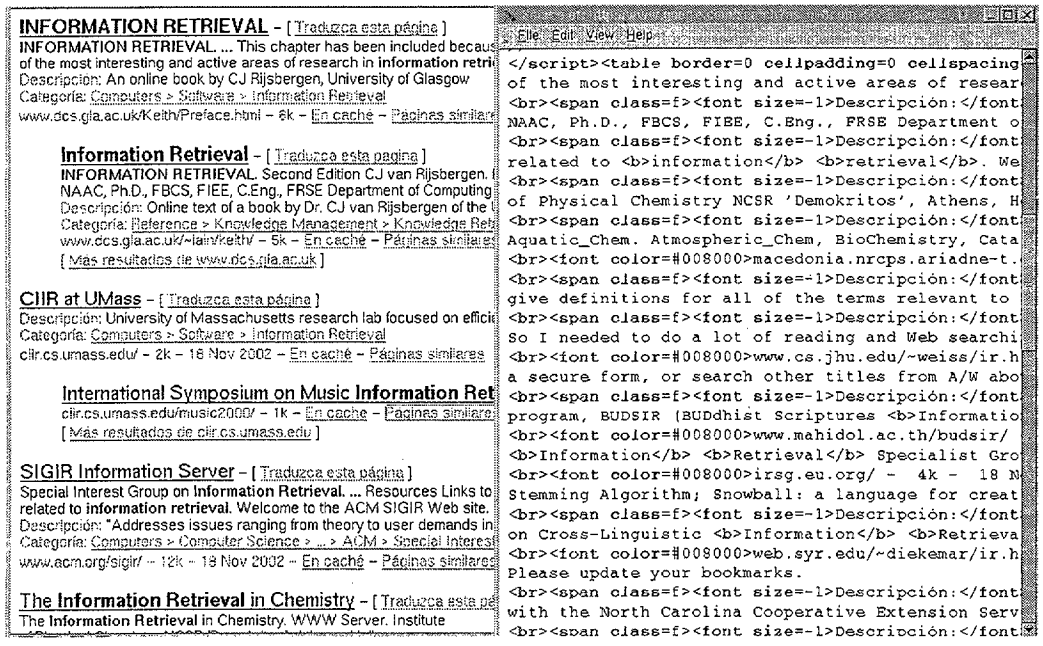


Figura 1.3: Regularidad en el Web

pero puede ocurrir que algunos pocos de esos artículos vengan acompañados además de un indicativo promocional de oferta, que otros artículos no poseen. Es precisamente la posibilidad que HTML ofrece para realizar múltiples y flexibles combinaciones de marcado de sus etiquetas sin apenas restricciones, una de las razones por las que el Web de hoy en día resulta tan heterogéneo a simple vista, pese a que los principios de su funcionamiento sean en realidad sencillos. Es precisamente esa combinación de sencillez y flexibilidad, carentes en el mundo de las bases de datos, una de las principales razones que han encumbrado a HTML como el formato por excelencia de los documentos en el Web.

Reconocer esta regularidad estructural en una página permite extraer sus datos relevantes. Existe una gran necesidad actualmente de aplicaciones que puedan integrar la información semiestructurada de varias fuentes diversas preexistentes. La heterogeneidad de la forma de marcar datos en cada fuente hace necesario muchas veces un pretratamiento particularizado consistente en adaptar los datos provenientes de diversas fuentes a un formato común. Una vez estructurados todos los datos en un formato común, éstos pueden ser procesados adecuadamente sin tener en cuenta su procedencia. Esto suele ser necesario en procesos de integración de información proveniente de distintos servidores cuya forma de acceso no es posible que sea modificada. Ello

suele ocurrir muchas veces en el Web en general, o, a menor escala, en los servidores localizados en la intranet de una empresa, donde es primordial que los sistemas sigan funcionando para las personas y herramientas que los llevan usando sin necesidad de ser interferidos por el hecho de que una nueva herramienta de acceso se incorpore al sistema informático preexistente.

1.4.6. Ausencia de semántica en el marcado

Cualquier dato que aparezca en una página HTML no puede distinguirse fácilmente de forma automatizada. Por ejemplo, un dato numérico, no puede reconocerse fácilmente como un precio, una cantidad, una fecha o un número de referencia por la sencilla razón de que en el marcado HTML del Web actual no se distingue entre unos tipos de datos respecto de otros. Por otro lado, prácticamente casi cualquier combinación de etiquetas HTML puede usarse indistintamente para marcar unos datos u otros, razón por la que acaba siendo el usuario quien, aplicando sus conocimientos contextuales en la navegación, infiere la semántica de esos datos al visualizarlos en la pantalla. Esto es algo que ocurre cuando se navega con un browser. En la navegación manual se debe tomar muchas veces la decisión de pulsar en uno u otro enlace o la de rellenar un formulario en lugar de otro, o la de rellenar cierto campo de una forma en lugar de cierta otra dependiendo de los datos visualizados hasta el momento en la pantalla, pero que el ordenador no es capaz de distinguir respecto del resto de los datos visualizados porque HTML es un lenguaje de marcado poco orientado a la descripción semántica y con un uso muy orientado a la visualización.

1.4.7. Niveles de estructuración

Tradicionalmente, en el campo de la documentación, se ha distinguido entre datos estructurados y datos no estructurados. Los datos no estructurados son aquellos que no presentan ningún esquema o estructura más allá de la mera secuencia de bytes o palabras. Los datos estructurados son aquellos que siguen un esquema de datos perfectamente definido (e.g. aquellos contenidos en bases de datos). El esquema que define la estructura de un conjunto de datos estructurados suele incluir férreas restricciones sobre los mismos, tales como un fuerte tipado de datos, restricciones en los rangos posibles de valores admitidos para ciertos datos, unicidad, cardinalidad de las repeticiones, limitaciones en el tamaño o posibilidad de impedir la existencia de campos con valores nulos.

Datos no estructurados

Un ejemplo típico de datos no estructurados son los textos libres. Debido a su escasa o prácticamente nula estructura, la única manera posible de recuperar información de estos documentos es mediante consultas no estructuradas o imprecisas, tales como las búsquedas por palabra clave. Con este tipo de consultas sólo es posible representar expresiones que recuperen aquellos documentos en los que aparece el conjunto de palabras clave buscadas. Es habitual en este tipo de consultas la posibilidad de utilizar operadores lógicos para combinar expresiones simples. Básicamente, éste es el sistema de datos en el que se basan los buscadores de Internet como Google o Altavista.

Si bien este tipo de consultas son sencillas de realizar y existe un gran número de trabajos e importantes contribuciones a este respecto [28], lo habitual es que este tipo de soluciones trabajen con un nivel de granularidad demasiado grueso (el documento), frente a lo que muchas veces en realidad se necesita (el dato). Los muy bien conocidos buscadores como Google o Altavista consideran el Web como un gran repositorio de información no estructurada y son capaces de construir gigantescos índices sobre el Web, permitiendo su consulta de forma eficiente. Sin embargo, para un cada vez mayor número de usuarios que encuentran la necesidad de realizar consultas más especializadas, esta abstracción del Web como un conjunto de información no estructurada es demasiado débil, ya que, a fin de cuentas, no sirven para nada más que para localizar documentos que tengan las palabras consultadas, no para obtener el dato en sí que en realidad buscan, ni tampoco para automatizar el uso de aplicaciones manejables desde formularios.

Datos estructurados

Los datos estructurados son aquellos que presentan un esquema rígido y bien definido para los datos. Un ejemplo típico de fuente estructurada de datos son las bases de datos relacionales. En este caso existe un diccionario de datos que define la organización interna y las restricciones de los datos, así como de las relaciones entre los mismos. Cuando los datos se encuentran en forma estructurada, es posible realizar consultas precisas mediante lenguajes de consulta estructurados, de los cuales el más popular es SQL [78]. Esta precisión a la hora de hacer consultas hace que los datos estructurados sean muy adecuados para su tratamiento por programas de ordenador.

Datos semiestructurados

Al contrario de los datos estructurados, los datos del Web carecen de todas estas restricciones y pertenecen a una categoría distinta. Esa categoría de datos ha recibido el nombre de datos semi-estructurados [36]. Los datos semi-estructurados se caracterizan porque, aunque siguen algún tipo de esquema, el seguimiento que hacen del mismo es mucho menos rígido que en el caso de los datos estructurados. Según [60], las principales características de los datos semiestructurados son:

- El esquema de datos no es explícito, esto es, no es conocido de antemano. Puede existir, pero estará, en todo caso, implícito en los datos y cualquier restricción asumible de ese esquema deberá inferirse de ellos de alguna manera.
- El esquema de datos está sujeto a cambios y puede cambiar con frecuencia. No existen impedimentos para que esto pueda ocurrir en cualquier momento.
- El esquema tolera irregularidades que no siempre aparecen especificadas.
- No existe un tipado fuerte de los datos individuales, por lo que es posible encontrar en ocasiones valores de tipos diferentes a los esperados. Por ejemplo, para un mismo tipo de dato consultado, es posible que éste aparezca en varios formatos distintos.

Sin tener la programabilidad de una férrea base de datos en la que es fácil asumir decisiones semánticas acerca de los datos, su bajo coste de creación las ha convertido en poco tiempo en la alternativa usada por excelencia para la publicación de datos en el Web. Sin embargo, la gran mayoría de la información accesible por el Web no presenta un nivel de estructuración tan fuerte, pese a que sería deseable poder manipularla eficazmente. En muchas ocasiones, especialmente en el caso de bases de datos accesibles desde el Web, la estructura de marcado HTML en la que esos datos se ven incrustados hace perder fácilmente la visión de esa estructura férrea. HTML es un buen formato para ser visualizado en navegadores. Sin embargo, su estructura no está orientada a la descripción de los datos, sino a su visualización. Téngase en cuenta, que en el proceso de transformación de la información desde repositorios estructurados a formatos visualizables, se produce una pérdida de estructuración que, a lo largo de los últimos años, las diversas aplicaciones que volcaban al Web los contenidos de las bases de datos, han venido

produciendo sin demasiados reparos. Es ahora, con un Web lleno de contenidos inmanejables eficazmente, cuando los métodos para reestructurar la información del Web son más necesarios.

HTML es el lenguaje de marcado en el que se encuentran embebidos mayoritariamente los datos del Web. Apenas los *Web Services* [53] y unos pocos y marginales (comparativamente con el inmenso tamaño del Web) segmentos de negocio usan XML como formato de intercambio de datos. Algunos más son los sitios que aparecen publicados en XHTML [117]. Muy al contrario de los documentos XML, las páginas HTML se caracterizan porque su estructuración está atada a muy pocas reglas. La gran mayoría de ellas son reglas elementales de construcción, sin que las reglas lógicas de estructuración propias del ámbito de conocimiento del documento queden bien reflejadas en su marcado estructural. Este marcado, por el contrario, sí suele contener una elevada carga de aspectos orientados a la visualización. Esta débil estructuración de las páginas puede ser cambiada fácilmente en cualquier momento, pues no existe un esquema de datos explícito que deba regir la estructura de las páginas. Por todos estos motivos, los documentos HTML pueden ser considerados como datos semiestructurados.

La tabla 1.5 presenta un resumen de las principales diferencias entre las características de los datos según sus distintos niveles de estructuración.

	Datos no estructurados	**Datos estructurados**	**Datos semi estructurados**
Esquema de datos	Inexistente	Explícito	Implícito
Restricciones	Inexistentes	Férreas	Laxas
Irregularidades	Inexistentes	Prohibidas	Abundantes
Cambios de estructura	Inexistentes	Prohibidos	Habituales
Consultas	Imprecisas	Precisas	Poco precisas
Ejemplo	Texto plano	Base de datos	Página Web
Coste de creación	Bajo	Alto	Bajo
Uso	Medio	Bajo	Muy alto

Cuadro 1.5: Diferencias entre características de los datos según su nivel de estructuración

1.4.8. Distribución de la información

Los datos que aparecen en el Web y son necesarios para una tarea no suelen aparecer todos integrados en un mismo documento, sino que suelen

aparecer distribuidos en varios de ellos por cuestiones organizativas. Así pues, información que necesita ser combinada para realizar una tarea puede encontrarse distribuida de muy diversas formas:

- Entre los documentos de diversos sitios Web, como por ejemplo informaciones complementarias de restaurantes en distintos portales de ocio.
- Entre los documentos individuales integrantes de otros documentos multimedia, como por ejemplo las diversas páginas que formen un mismo `frameset`.
- Entre diversas páginas enlazadas entre sí, como por ejemplo, los derivados de la paginación de resultados de búsqueda.
- Mezclada dentro de una misma página con información no relevante para la tarea, como por ejemplo, la publicidad y otras informaciones no relevantes para la tarea.

1.4.9. Difícil modificabilidad

Los programas creados para navegar automáticamente en el Web, deben, por lo tanto, afrontar el problema del manejo de información semiestructurada, corriendo el peligro de ver modificada, sin previo aviso, la estructura de las páginas que se desea visitar. En el caso de la integración de aplicaciones accesibles desde el Web, éstas deben ser tratadas con sumo cuidado y respeto para no interferir en el correcto funcionamiento de las tareas realizadas por otros mecanismos, típicamente browsers, que otros usuarios estén llevando a cabo. La mejor forma de integrar estas aplicaciones *legadas* suele consistir, no en modificarlas para que contemplen una nueva herramienta de acceso, sino en usar las interfaces que ya tengan desarrolladas, típicamente formularios en formato HTML, creando herramientas que emulen las herramientas de acceso preexistentes que hasta el momento han estado siendo usadas. En muchas ocasiones ello supone, si no un serio ahorro de costes, la única opción aplicable.

1.4.10. Aportaciones de XML

XML [132], concebido como una tecnología para definir lenguajes de marcado propios para cada campo del conocimiento, aporta numerosas y significativas soluciones a la hora de estructurar los documentos de una forma

mucho más clara que la manera en la que HTML lo permite actualmente. Gracias a su mayor expresividad, y a su facilidad para ser procesado por programas no orientados a la mera visualización, el uso de XML se ha extendido rápidamente como solución al intercambio de datos estructurados en muy diversos ámbitos. Las principales cualidades de los documentos XML son las siguientes:

Sintaxis fácilmente procesable

XML presenta una sintaxis textual muy simplificada que permite estructurar fácilmente los documentos en forma de árbol. Al contrario de su predecesor SGML [65], los programas que procesan XML pueden ser muy fácilmente construibles debido a la simplicidad del formato, algo que sin embargo no le resta flexibilidad y extensibilidad para poder definir documentos complejos.

Independencia de la presentación

XML permite dotar fácilmente a los documentos de una estructura sintáctica que esté adecuada a la naturaleza propia del documento, prescindiendo absolutamente de la forma en la que éste pueda ser visualizado, labor que queda delegada en las hojas de estilo.

Estructuración de datos

Las etiquetas XML no definen propiedades acerca de cómo deben ser visualizadas, sino que simplemente describen los datos que contienen. Con el fin de dar una descripción formal a los posibles contenidos que puede tener una etiqueta dentro de un documento, XML incluye la posibilidad opcional de describir esas sintaxis mediante los DTD o XML Schema.

Contexto semántico

Algo que no forma parte de la especificación de XML y para lo que aún no se ha definido formato estandarizado de representación es el **contexto semántico**. Mediante un contexto semántico capaz de dotar de significados a cada una de las partes de esa estructuración sintáctica (etiquetas y atributos XML), es posible así que cada uno de los datos que aparecen en un

documento pueda tener significado semántico para los usuarios. De esta forma, las distintas partes de los documentos pueden presentar un significado conocido y sin ambigüedades, adecuado para ser entendible por las máquinas, algo que las páginas HTML que forman parte del Web actual no cumplen, pues están orientadas a la mera visualización. La tabla 1.6 refleja un resumen de las aportaciones que realiza cada una de las tecnologías mencionadas en este párrafo a los documentos estructurados en XML.

Tecnología	Aporta	Finalidad	Herramientas
XML	Sintaxis	Datos descritos	DTD, Schema, Relax-NG, ...
Hojas de estilo	Presentación	Documentos presentables	CSS, XSL, ...
Contexto semántico	Semántica	Documentos procesables	Metadatos RDF/OWL, programas ...

Cuadro 1.6: Resumen de aportaciones de XML

Sin embargo, siendo XML una buena opción, lamentablemente aún no es usada masivamente en el Web. Para ello resulta necesario un cierto consenso a la hora de escoger el lenguaje concreto de entre los múltiples lenguajes existentes de un dominio de conocimiento para marcar adecuadamente los contenidos. Por otro lado, lo cierto es que XML apenas resuelve una parte del problema: la de la estructuración de los documentos y la posibilidad de asociar, gracias a un contexto que suele estar implícito en las personas, significados semánticos a cada una de esas partes estructuradas del documento. Sin embargo, XML no realiza aportaciones a la forma en la que esos documentos deben ser obtenidos de las fuentes de datos, algo que normalmente no se puede conseguir en un solo paso.

Sin embargo, al igual que ocurre con la orientación a la accesibilidad e independencia del dispositivo de acceso, son numerosos los inconvenientes que permiten augurar que XML aún tardará mucho tiempo en ser una solución efectiva. Entre esos inconvenientes cabe destacar los siguientes:

- El uso de XML directamente en los browsers, como mero formato de visualización, acompañado, eso sí, de sus correspondientes hojas de estilo, no aporta ventajas sobre otros formatos para los que ya existen browsers, como HTML.
- El ya gigantesco tamaño del Web supone sin duda una gran inercia a la hora de acoger nuevas tecnologías. Ello implica que, aunque en determinado momento, XML pase por fin a ser la solución comúnmente

aceptada por los servidores Web, la lentitud con la que se reajustarán las páginas de los servidores Web ya existentes implicará que el acceso a sitios Web basados en HTML seguirá siendo necesario durante mucho tiempo. Y aún así, no es previsible tampoco que HTML llegue a desaparecer por XML, por lo que habrá un gran número de aplicaciones que no migrarán a XML. En cualquier caso, en la actualidad, tras cuatro años después del nacimiento de XML, éste apenas se encuentra usado en unos pocos segmentos verticales de negocio, de una forma marginal si se lo compara con HTML.

- El hecho de que las especificaciones del W3C que parecen prometer nuevas soluciones estén aún en fase de borrador supone un serio inconveniente para ser adoptadas por los desarrolladores, que las contemplan aún con muchas reticencias.

1.5. Coste de la navegación automatizada

Uno de los aspectos más curiosos para la automatización de tareas en el Web es que la mayoría de las mejoras de accesibilidad mencionadas en el apartado 1.4.3 en lo referente a la independencia de dispositivos de acceso en la navegación con browsers, son también directamente aprovechables para otras aplicaciones que no son browsers, como las de navegación automática. Un marcado poco orientado a los aspectos de visualización, que delegue esa labor en hojas de estilo, independiente del dispositivo de acceso, y cercano a la estructura lógica del documento, es más simple y regular y menos proclive a ser modificado con el tiempo, pues serán las hojas de estilo quienes asuman los cambios de presentación. Ello redunda en una importante minimización de costes, tanto de publicación en el lado del servidor como de procesamiento en el cliente que manipule las páginas obtenidas del servidor.

En el lado del servidor, la opción de mantener documentos accesibles, cada uno de ellos visualizable con varias hojas de estilo alternativas, cada una de ellas a su vez orientada a un dispositivo de acceso, será sin duda mucho más económica (en tiempo, espacio y esfuerzo a largo plazo) y escalable que la de mantener para cada página una versión orientada a cada terminal.

En el lado del cliente, la simplicidad del marcado de las páginas accesibles reduce sensiblemente la complejidad de procesamiento. Esto tiene especial importancia en las reglas de extracción de datos basadas en la regularidad estructural de las páginas. Al ser esta regularidad estructural más simple y más estable, estas reglas serán a su vez más simples y estables, por lo que

necesitarán un menor esfuerzo de desarrollo y de mantenimiento.

Las aplicaciones capaces de navegar en el Web que aparecen en el apartado 1.3 pueden, sin duda, ser construidas desde varias plataformas basadas en distintos lenguajes de programación. Desde C, Java, Perl, TCL-TK, Prolog, Visual Basic hasta el mismo ensamblador, cualquiera de estos lenguajes puede ser usado en la construcción de estos sistemas, habida cuenta de la existencia de varias bibliotecas de soporte utilizables en cada uno de estos lenguajes, como las mencionadas en el apartado 3.4. Sin embargo, no todas las alternativas presentan la misma flexibilidad para cada labor, ni tampoco tienen los mismos costes, soliendo haber unos lenguajes más orientados que otros para cada tipo de labor. Para minimizar los costes, es necesario que la plataforma de ejecución proporcione un buen soporte al acceso de datos en el Web por el protocolo HTTP, emulando lo más parecidamente el comportamiento de un browser y ejecutando la mayor cantidad de acciones que implícitamente éste ejecuta, algunas de las cuales aparecen en el apartado 2.1, como por ejemplo que éste sea robusto en los fallos que ocasionalmente ocurren en las comunicaciones (sobrecarga en la conexión, tiempo de respuesta excesivo en el servidor, ...), enlaces rotos a componentes del documento, y que permita ubicar fácilmente los errores cuando éstos se produzcan, para ayudar a una más rápida reparación del código de la aplicación. Desde el punto de vista del programador, la plataforma de ejecución debe proporcionarle además un API de desarrollo con el adecuado nivel de abstracción, la posibilidad de añadir sus propias medidas de robustez ante inesperados comportamientos del servidor (páginas de error, cambios de regularidad estructural, ...) y, sobre todo, una sencilla forma de definir para fuentes de datos semiestructuradas, reglas de extracción de datos, sencillas, potentes y fáciles de mantener, pues suelen ser la parte que más suele sufrir ante los inevitables cambios en la regularidad estructural de las páginas. Los costes de la navegación automatizada, que aparecen resumidos en la tabla 1.7, pueden separarse en tres grandes grupos:

Coste	Necesidad	Para programador	Para plataforma
De desarrollo	Buen API	Buen nivel de abstracción	Buen soporte acciones implícitas
De ejecución fallida	Robustez	Ante cambios o fallos del servidor	En comunicaciones
De mantenimiento	Bajo coste	Reglas de extracción de datos	Ubicación de errores

Cuadro 1.7: Resumen de tipos de coste de la navegación automatizada

1.5.1. Coste de desarrollo

El desarrollo de aplicaciones de navegación automatizada debe tener en cuenta las características de los datos que va a manejar, que se encuentran en el apartado 1.4. El coste de desarrollo depende sin duda de la complejidad de la tarea que se desea automatizar, pero también es altamente dependiente del API que ofrece la plataforma de desarrollo de estos programas. En general, lo deseable es que este API tenga un completo soporte de las acciones básicas implícitas que aparecen en el apartado 2.1 y que a su vez ese API tenga un buen nivel de abstracción para poder facilitar convenientemente la implementación de las acciones básicas explícitas, esto es, los pasos básicos de la tarea, detallados en el apartado 2.2. Una óptima plataforma de desarrollo será aquella que permita detallar cada una de estas acciones con el mínimo número de líneas de código de alto nivel, entendibles por mucha gente y evitando en la medida de lo posible que sus diferencias con un browser trasciendan al usuario, para garantizar que la parte de la tarea que especifica la recuperación de documentos del Web sea realizada de la forma más sencilla posible.

Sin embargo, la recuperación de documentos del Web no es más que una de las acciones que deben formar parte de la tarea. Tal y como figura en el apartado 2.2, debe procesarse el documento para extraer los datos relevantes que en él figuran, estructurarlos, opcionalmente homogeneizarlos en el caso de que puedan proceder de varias fuentes y procesarlos conforme a la tarea que se esté automatizando. Todas estas acciones deben ser programadas para cada tarea. Sin duda, dado que a fin de cuentas el desarrollo de estos programas es una labor de programación, conviene dotar al programador de las mejores técnicas de reutilización de código propias de los lenguajes de alto nivel de abstracción, como son la encapsulación en funciones definidas por el usuario y la modularización de las mismas para que pueda construir sus propias bibliotecas reutilizables por él y por otros usuarios. De esta forma puede reducirse igualmente el *time to market*, es decir, el tiempo de desarrollo y se puede disfrutar más fácilmente de prototipos más rápidamente aplicables.

No obstante, una de las principales fuentes de coste de desarrollo de estos programas es la falta de soporte adecuado para muchas acciones ocultas al usuario durante la navegación y que son normalmente ejecutadas por los browsers. Siendo lo ideal que dichas acciones sean llevadas a cabo de forma transparente por estas plataformas para que no trasciendan al usuario, lo habitual es que las actuales plataformas de desarrollo de estos programas proporcionen un soporte bastante incompleto de las mismas, por lo que al final, el programador debe introducir su propio código para llevar a cabo un

sinfín de pequeños detalles técnicos de los que los usuarios de los browsers no acaban siendo normalmente conscientes y que aumentan considerablemente la complejidad de la tarea. De esta forma, una pequeña y sencilla labor en el Web puede fácilmente requerir el desarrollo de programas demasiado grandes y complejos que intentan resolver una gran variedad de pequeños detalles técnicos más propios de la configuración del servidor que de la propia tarea y que además, al ser tremendamente cambiantes, suponen una importante fuente de futuros fallos de ejecución.

El coste de desarrollo queda minimizado con la utilización de lenguajes de programación con el adecuado nivel de abstracción y soporte transparente a los detalles de la navegación. Un lenguaje de programación de alto nivel de abstracción, encapsulable, modular, con la posibilidad de definir fácilmente reglas de extracción de datos basadas en la regularidad estructural y con flexibilidad para estructurar esos datos extraídos resulta primordial para reducir los costes de desarrollo de estas aplicaciones.

1.5.2. Coste de ejecución fallida

El coste de ejecución fallida es el coste que supone un fallo de ejecución al realizar una tarea. Dicho coste suele depender de la relevancia de la tarea y no ser dependiente de la forma en la que este programa pueda haber sido construido. Este coste, no obstante, puede quedar convenientemente minimizado con la utilización de medidas capaces de dotar de robustez a la navegación, para que el programa trate de la forma más adecuada los comportamientos no esperados de las comunicaciones o del servidor. Básicamente, para aplicar las medidas de robustez se suele tener en cuenta la ubicación del fallo y el tipo de tratamiento que se desea aplicar.

Ubicación del fallo

Fallos en las comunicaciones Las comunicaciones con el protocolo HTTP pueden fallar si existe una sobrecarga en la conexión TCP/IP, el servidor al que se intenta conectar está inaccesible o su respuesta tarda un tiempo considerado demasiado alto. Este tipo de fallos, normalmente notificados al programador en forma de excepciones o de códigos de error, pueden ser convenientemente tratados tanto desde el código de las bibliotecas de la plataforma de ejecución, como por código del usuario.

Fallos del servidor Las aplicaciones del Web *legado* no siempre funcionan

de la manera en la que de ellas se espera. Ante correctas peticiones de procesamiento de formularios, las aplicaciones del Web pueden funcionar puntualmente de forma incorrecta, devolviendo al cliente páginas de error inesperadas por éste. En estos casos, el error no está ni en la aplicación de navegación, ni en la conexión TCP/IP, sino en el servidor. Distinguir una página de error (que normalmente ha sido obtenida sin errores desde el punto de vista del protocolo HTTP), de una página que confirme que se han procesado adecuadamente los datos enviados en un formulario, no es algo realizable de forma genérica, sino que dependerá de la aplicación y de su forma de devolver páginas de error. Aunque otras medidas son posibles, lo habitual suele ser tratar estos errores insertando código que *asevere* que la respuesta de una petición indique que ésta ha sido correctamente procesada, típicamente con una sentencia del tipo *assert*. No insertar este tipo de comprobaciones puede impedir el tener garantizado que el servidor haya procesado adecuadamente la petición recibida. Por otro lado, la página devuelta puede no ser una página de error y contener los datos que espera el cliente, pero puede estar tan mal construida, que no se respeten algunas normas básicas de construcción de páginas necesarias para su correcto procesamiento. En esos casos, lo habitual suele ser corregir internamente los errores de estas páginas antes de que sean manipuladas por el programa para que presenten ante el programador una estructura correctamente procesable.

Fallos del cliente por cambios en el servidor Por otro lado, conforme a lo expuesto en el apartado 1.4.7, la regularidad estructural de las páginas del servidor puede cambiar en cualquier momento al ser el Web una fuente semiestructurada de datos. Por este motivo, los programas de navegación en el Web suelen incluir código para detectar los cambios de regularidad estructural en las páginas visitadas, de forma que, si ésta cambia, el programa pueda actuar en consecuencia. Lo habitual suele ser capturar el error y ejecutar reglas secundarias de extracción de datos, o bien no capturar el error, propagando excepciones y confiando en que la regularidad estructural no cambie durante mucho tiempo. Dependiendo del grado de robustez que se quiera añadir a estos programas y del coste asumible por el programador, se puede insertar un número mayor o menor de reglas secundarias que redunden en una mayor posibilidad de encontrar el dato buscado ante estos posibles cambios.

Naturaleza del tratamiento

Tratamiento genérico Los tratamientos genéricos son aquellos que pueden implementarse para solucionar errores producidos en la navegación con cualquier servidor o aplicación accesible desde el Web. Se trata de fallos que son procesables en las rutinas de las bibliotecas de la plataformas de ejecución y que suelen permitir cierta parametrización desde el código del cliente. Ejemplos de estos tratamientos genéricos son algunas rutinas existentes, como [104, 58, 146], que permiten reparar la estructura de las páginas HTML mal construidas. Otros ejemplos son los temporizadores, que limitan el tiempo máximo admisible en el que debe ser resuelta una petición en el Web. También puede llevarse a cabo una política de **reintentos suaves**, consistente en reintentar varias veces una petición HTTP que no tenga efectos colaterales hasta que sea correctamente procesada, quizá con un número máximo de intentos. Se denomina petición sin efecto colateral a la que no modifica datos esenciales en el servidor y puede ser realizada múltiples veces sin temor a sufrir efectos colaterales. Una petición con efectos colaterales, como por ejemplo una petición de transferencia bancaria, no debe ser reintentada múltiples veces, salvo que se asuma el coste de su multiplicidad. Algunas iniciativas como [114] proponen extensiones al protocolo HTTP para que las peticiones con efectos colaterales puedan ser realizadas completamente de forma atómica, o que, en caso de fallo, no tengan efectos colaterales, pero se trata aún de una iniciativa sin estandarizar y carente de soporte en el Web actual. Si todo esto falla, el usuario aún puede tratar el error en su código capturando las excepciones adecuadas o analizando los errores de estas llamadas con el fin de aplicar un tratamiento particularizado.

Tratamiento particularizado Los tratamientos particularizados son aquellos que no son aplicables de forma genérica en bibliotecas, sino que, al depender de las aplicaciones concretas que estén involucradas en la tarea, son mejor tratables en el código del programador (por lo que su coste es también más elevado). Son múltiples las acciones de tratamiento que puede emprender un programador. Por ejemplo, puede aplicar una política de reintentos ante peticiones no correctamente respondidas, como por ejemplo, cuando el servidor indica que está sobrecargado de trabajo y responde con una página en la que indica que se reintente la petición más adelante. También es posible inundar el programa con reglas de extracción alternativas y redundantes para que se pongan en ejecución en el caso de fallos de las reglas principales de

extracción de datos. También es posible insertar código para asegurarse de que cada petición ha sido respondida correctamente y lanzar un mensaje de error y aviso al programador cuando no sea así. En la tabla 1.8 aparece un resumen de las medidas de robustez más habituales aplicadas según el origen del fallo.

	Síntoma	Tratamiento genérico	Tratamiento particular
Lugar	Varios	Plataforma de ejecución	Código de programador
En comunicaciones	Excepciones, errores	Temporizadores, reintentos suaves	Reintentos
Fallos del servidor	Páginas de error	Reparación de páginas	Asserts, reintentos
Cambios de estructura	Fallos en las reglas	No	Reglas secundarias, asserts

Cuadro 1.8: Resumen de medidas de robustez según origen del fallo

1.5.3. Coste de mantenimiento

Los programas de navegación genérica no adaptada apenas necesitan labor de mantenimiento. Ésta está apenas centrada en algunos browsers, para los que frecuentemente se lanzan nuevas versiones que mejoran imperfecciones de sus predecesoras o mejoran alguna pequeña funcionalidad. Las modificaciones en la regularidad estructural de las páginas no las afecta, como sí lo hace a las aplicaciones de navegación genérica adaptada y particularizada. En el caso del Web Semántico, por ejemplo, esas modificaciones deben quedar convenientemente reflejadas en los metadatos del sitio Web, de forma que el mantenimiento se reduce a una labor declarativa. Sin embargo, en los programas de navegación particularizada, este mantenimiento es una labor muy costosa que debe realizarse habitualmente en el mismo código del programa. Por muchas medidas de robustez que se quieran añadir para alargar la vida del programa, inevitablemente algún cambio no contemplado en la regularidad estructural acabará apareciendo tarde o temprano, invalidando alguna de las suposiciones asumidas por el programador. La labor de mantenimiento es en estos casos la única opción. Con el fin de minimizar costes, suele ser deseable que el mantenimiento lo pueda realizar personal que no necesariamente tenga una elevada preparación. Es por ello que resulta tremendamente importante que los programas de navegación particularizada cumplan tres importantes requisitos: legibilidad, brevedad y simplicidad.

Legibilidad

Las personas que realizan el mantenimiento de los programas no tienen por qué ser necesariamente las mismas que participaron en su desarrollo. Incluso aún cuando sí sean las mismas personas, las medidas de robustez anteriormente mencionadas pueden haber sido efectivas durante bastante tiempo y el programador que quiera realizar una labor de mantenimiento puede haber olvidado los detalles en los que se basó para su construcción. Por todas estas razones y por muchas otras, la legibilidad de los programas resulta primordial. Para mejorar esa legibilidad, es conveniente usar técnicas de programación con un nivel de abstracción adecuado para la descripción de tareas y que preferiblemente sean conocidas por mucha gente y estén basadas en estándares que eviten la ambigüedad, de forma que cualquier pequeño trozo de código pueda ser fácilmente entendido por las personas que lo analicen sin necesidad de tener un conocimiento completo del problema.

Brevedad

La legibilidad no es el único factor que influye en el coste de mantenimiento de estos programas. Un wrapper escrito en C o Java puede ser fácilmente legible al estar escrito en un lenguaje de programación conocido por mucha gente, pero si no usa unas bibliotecas de soporte con el adecuado nivel de abstracción, con primitivas que reflejen cada una de las acciones de la tarea en pocas líneas de código, el tamaño del programa final puede fácilmente dispararse. Para conseguir programas breves, lo deseable es poder disponer de un potente conjunto de primitivas capaces de reflejar cada paso de la tarea en unas pocas líneas de código.

Simplicidad

Una buena plataforma de desarrollo se caracteriza por la flexibilidad y potencia de sus primitivas de construcción, las cuales deben ser capaces de permitir a su vez la suficiente parametrización para poder controlar todos los aspectos técnicos de bajo nivel que requieran ser atendidos. Sin embargo, esa potencia suele conseguirse implicando cierta complejidad de manejo por parte del programador, por lo que es deseable además de las propiedades anteriores de legibilidad y brevedad, la suficiente simplicidad del programa para que cualquier cambio que se realice en el mismo no acabe afectando a demasiadas líneas de código. Por esa razón, es deseable usar mecanismos que, permitiendo la programación de complejos algoritmos de navegación y

manipulación de datos, mantengan lo más simple posible los programas con el objetivo de que sea poco costosa la labor de localizar y modificar las líneas de código implicadas en un cambio dentro de uno de estos programas. La simplicidad de mantenimiento se refleja en la capacidad de que cualquier cambio pueda fácilmente localizarse sobre una pequeña parte del programa (una línea de código es lo ideal) sin que el cambio implique necesariamente una revisión del resto del programa. Es importante que la plataforma de ejecución pueda indicar el lugar donde se producen los fallos con el fin de tener localizado el lugar en el que debe aplicarse un cambio.

1.6. Marco de trabajo

Numerosos han sido los trabajos que han elaborado aplicaciones de navegación particularizada para los sitios Web en los últimos años. Buena parte de ellos aparece resumida en el capítulo 3. Todos ellos coinciden en presentar soluciones con serios **costes de desarrollo**, problemas de **aplicabilidad** en diversas fuentes de datos (dadas sus enormes heterogeneidades), una elevada **fragilidad** ante errores y cambios en el servidor, y, principalmente, elevados costes de **mantenimiento** que sin duda han influido en su escasa utilización.

Por estas razones, nuevas formas avanzadas de recuperación y tratamiento de la información para la navegación Web particularizada, aplicable al *deep Web* que se sabe localizada en el Web legado, con las características mencionadas en el apartado 1.4, necesitan ser aplicadas, de forma que se aprovechen los avances de los últimos estándares orientados al Web, con el fin de dar soluciones efectivas a los problemas anteriormente mencionados minimizando adecuadamente sus costes.

1.7. Objetivos

En la presente tesis doctoral se pretende aportar soluciones para los problemas mencionados en el apartado 1.5 y que se encuentran resumidos en el apartado 1.6. Con el fin de facilitar la automatización de tareas en el Web legado, tanto basado en HTML como en cualquier formato XML, se persiguen métodos avanzados, para reducir el coste de desarrollo y mantenimiento de aplicaciones de automatización, de forma que a su vez éstas sean lo más robustas posibles ante posibles errores en las aplicaciones que ejecutan en los servidores y ante la orientación a la visualización de las páginas y a sus

cambios de regularidad estructural. A continuación se detallan los objetivos de esta tesis:

1. Proponer unos **mecanismos de desarrollo** de programas que naveguen automáticamente en el *deep Web*, de forma que el coste de desarrollo de estas aplicaciones, mencionado en el apartado 1.5.1, se vea sensiblemente reducido frente a las opciones existentes actualmente y la automatización de tareas en el Web por medio de programas de navegación particularizada sea así asequible a un mayor número de personas. Estos métodos de construcción deberán tener un adecuado nivel de abstracción, con el fin de favorecer su comprensión, y, en lo posible, permitir su utilización por herramientas que ayuden al programador en su desarrollo.

2. Proporcionar una **plataforma** que no sólo pueda usarse con un elevado nivel de abstracción y comprensible por los usuarios conforme al objetivo anterior, sino que además sea convenientemente parametrizable y con el mayor cubrimiento posible a la ejecución de aquellas acciones que implícitamente el browser realiza de forma transparente evitando en lo posible que el usuario sea consciente de ello. Dichas acciones, que aparecen detalladas en el apartado 2.1, son necesarias para mantener correctamente el diálogo y el concepto de sesión con los servidores Web con los que se dialoga. El fin de esta mejora consiste en abstraer al usuario del mayor número posible de detalles de bajo nivel, permitiéndole, sin embargo, tener el control de los mismos cada vez que lo necesite. Consiguiendo que la gestión de estas acciones genéricas de cualquier sitio Web queden convenientemente encapsuladas y no formen parte del código principal de los programas, los programas se mantienen lo más **compactos** posible.

3. Si bien es recomendable que la plataforma de navegación se haga cargo de las acciones de las que no es normalmente consciente un usuario que navega manualmente usando un browser, las **acciones que sí trascienden al usuario**, y que aparecen en el apartado 2.2, como seguir un enlace o rellenar un formulario, deben poder ser activables de la forma más sencilla posible por parte de éste. Lo ideal es que cada una de estas acciones pueda ser especificada con una acción sencilla en el caso de que use una herramienta (como por ejemplo en un navegador), o con un mínimo número de líneas de código, en el caso en el que emplee un lenguaje de programación, usando a ser posible un **conjunto pequeño pero potente de constructores flexiblemente parametrizables.**

4. Por otro lado, para **las acciones más complejas** que debe especificar el usuario, que se encuentran detalladas en el apartado 2.2 y que normalmente se corresponden con las acciones de procesamiento de datos mencionado en el apartado 2.2.6, puede requerirse la especificación de algoritmos complicados. Por ese motivo puede ser necesario la programación de varias líneas de código, con sentencias de programación como llamadas a funciones que manipulen complejas estructuras de datos, diversas condiciones que impliquen varios posibles casos alternativos a tener en cuenta, bucles que se aniden entre sí o con las sentencias condicionales y datos de diversos tipos. Lo habitual es que el código que procese los datos de una tarea sea muy particular y dependiente de ésta y de la forma en la que el usuario desee realizarla (por ejemplo, seleccionar la mejor oferta de leche fresca de un catálogo según varios criterios, como marca, precio, tamaño del envase y gastos de envío). Por ello, lo más aconsejable consiste en permitir la programación de **rutinas definibles por el usuario** en algún lenguaje de programación ampliamente conocido, como Java, u otros lenguajes igualmente reconocidos, de forma que esa rutina pueda ser convenientemente invocada desde el código de programa. De esta forma, pasando por argumento a esa rutina los datos extraídos y estructurados de las páginas almacenados en repositorios programables (variables, ficheros, ...), la rutina puede manipular los datos obtenidos del Web.

5. Proporcionar mecanismos adecuados para **favorecer la robustez** de los programas ante fallos en las conexiones TCP/IP y adaptabilidad ante cambios en las páginas, reduciendo el coste de ejecución fallida y que aparece detallado en el apartado 1.5.2. El objetivo consiste en incluir medidas de robustez ante fallos en las comunicaciones y ante cambios y fallos en las aplicaciones accedidas por el Web. Estas medidas se plasmarán tanto en la plataforma de soporte a la ejecución como en constructores directamente utilizables por el usuario en su propio código. Dicha funcionalidad no aparece en los browsers actuales, al igual que tampoco es común encontrarla en otros tipos de plataformas. También se debe contemplar una política de alternativas de uso en el caso de que se encuentren enlaces rotos.

6. Las medidas de robustez pueden amortiguar los efectos de los cambios de regularidad estructural del Web y pueden alargar el tiempo de vida de las aplicaciones de navegación automatizada. Sin embargo, debido al carácter siempre dinámico del Web, es necesario un **mantenimiento** para cuando esas medidas fallen. Por ello se pretende proporcionar

mecanismos para la minimización del coste de mantenimiento de las aplicaciones de navegación automatizada, que aparece detallado en el apartado 1.5.3, de forma que este mantenimiento quede reducido a una mínima labor, con un coste sensiblemente inferior a las alternativas utilizables hasta el momento, de forma que la mayoría de las acciones de mantenimiento consistan en la modificación de una acción sencilla, de las reflejadas en el apartado 2.2, fácilmente localizada y que normalmente afecte a muy pocas líneas de código del programa de usuario, sin ser necesario revisar el programa completo.

7. Permitir que esa automatización se pueda aplicar a prácticamente cualquier fuente o aplicación del **Web *legado***, es decir, a las tradicionales páginas HTML orientadas a la visualización y cuyos principales atributos aparecen detallados en el apartado 1.4 y que mayoritariamente abundan en el Web actual. La automatización de tareas en el Web se debe poder aplicar no sólo a las páginas que, por estar bien construidas, les sean inmediatamente aplicables las modernas tecnologías de manipulación de documentos y extracción de datos, sino también a aquellas que estén mal construidas (dentro de unos límites que las permitan ser convenientemente reparadas). Para ello, se incluirán en la plataforma de ejecución métodos capaces de reparar automáticamente esos errores de las páginas recuperadas de los servidores. Igualmente, deberán proporcionarse mecanismos, si no en la plataforma, al menos a disposición del usuario, para solventar los problemas derivados del hecho de que esas páginas estén pensadas exclusivamente para el acceso por medio de un conjunto limitado de browsers gráficos, o con capacidad de accionar eventos especiales desde rutinas embebidas en applets, animaciones Flash o, principalmente, rutinas JavaScript.

8. Permitir aplicar ese tratamiento automatizado de la información mencionado anteriormente también a documentos de **cualquier lenguaje definido sobre XML**, siempre que al menos su esquema sea conocible a priori por el programador para las tareas más complejas que así lo requieran. Así pues, se pretende que, aunque la mayor parte de las pruebas de implementación de esta tesis se realizará sobre páginas HTML del Web legado, ese formato concreto no deje de ser sólo un **ejemplo particular** de navegación sobre documentos en el Web, siendo posibles igualmente otros lenguajes definibles en XML, no ya sólo algunos de las ya conocidos (como WML + WMLScript [61], SMIL [125], RDF [123], RSS [16], SVG [124], MathML [120], ...), sino cualquier otro lenguaje XML en general, actual o futuro, es decir, ya inventado o aún por inven-

tar, independientemente de si su uso está centrado en la visualización en un browser o si tiene otra finalidad.

1.8. Estructura de la memoria

El capítulo 2 presenta un **análisis** de los principales aspectos de la automatización en el Web. Para poder realizar una efectiva automatización de tareas en el Web es preciso **analizarlas** primero, reconociendo sus partes o acciones básicas, examinando las alternativas aplicables a cada una de ellas, teniendo en cuenta las características distintivas de la materia tratada (los datos del Web). Para cada parte, se analizan sus principales ventajas e inconvenientes desde los puntos de vista de aplicabilidad y de coste de implantación, con el fin de poder escoger para cada problema la solución más adecuada. En el capítulo 3 se describe el estado del arte de la automatización de tareas en el Web, analizando desde sus precedentes hasta las tendencias actuales. Un estudio sobre las soluciones tecnológicas existentes, clasificadas según los criterios establecidos anteriormente puede encontrarse en el capítulo 4. Los capítulos 5 y 6 presentan unas propuestas de tecnologías para la **síntesis** de aplicaciones para la automatización de tareas en el Web que, basándose en las ideas del capítulo 2, presentan unos mecanismos de construcción de estas aplicaciones capaces de ser operables en el Web *legado* y que tienen además la aportación de estar basadas en estándares que reducen sensiblemente los costes de implantación y mantenimiento. El capítulo 7 introduce unos ejemplos que demuestran la aplicabilidad de estas propuestas a tareas definidas sobre sitios Web conocidos. Finalmente se enumeran algunas conclusiones del trabajo así como líneas de actuación para trabajos futuros.

Capítulo 2

Análisis de tareas Web

Cuando la gente navega por el Web, aparte de por ocio o entretenimiento, suele hacerlo persiguiendo un fin concreto, un propósito particular. Ejemplos de tareas Web quizá no tan complejas como las del apartado 1.2 pueden verse en la siguiente lista:

- Obtener un documento o un conjunto de ficheros multimedia
- Leer un periódico
- Usar un buscador (o varios)
- Enviar postales electrónicas
- Leer correo Web, lo cual implica las subtareas de identificarse, listar los mensajes, visitar siguientes, responder, ...
- Consultar disponibilidad de habitaciones en un hotel, suponiendo que ofrezca esta información online
- Comprobar que las páginas publicadas en un sitio Web cumplan ciertas propiedades (no tener enlaces rotos, adherirse a algún formato particular, ...)
- Comprar billetes de avión o de tren
- Comprar en una tienda electrónica, lo cual implica buscar cada artículo, añadirlo al carrito, estar atento a las ofertas, pagar, ...
- Reservar salas de reuniones en la intranet de una organización

- Presentar la declaración de la renta
- Poner una denuncia en la policía
- Y muchas más

Según crece el Web, con cada vez un mayor número de datos y aplicaciones accesibles, el número de tareas igualmente aumenta. De hecho, el crecimiento del Web no es el único impulsor del aumento de tareas realizables en el Web. Últimamente comienzan a ser cada vez más necesarias aplicaciones que combinen la información de varios sitios Web, de la misma forma en la que se combinan datos de bases de datos verticalmente fragmentadas y distribuidas [52]. Esas tareas empiezan a ser inmanejables para ser realizadas manualmente.

En otras ocasiones, por el contrario, sobre una misma aplicación accesible en el Web es deseable ejecutar varias tareas posibles. Pese a las reticencias de quien no quiere que sus datos sean comparados de forma explícita por un tercero, es poco extraño que un comparador de ofertas como [3] o de asociaciones de consumidores soliciten al Web de un banco la TAE de su mejor depósito bancario a seis meses para compararla con la de su competencia. De la misma forma, tampoco debería resultar extraño que ese mismo dato acabe siendo proporcionado a otra aplicación encargada de calcular el TAE medio ofrecido por los principales bancos españoles para un estudio estadístico. La utilidad que el dato tenga en la tarea es algo que depende mucho de la tarea y poco del servidor.

Para poder automatizar las tareas en el Web es preciso **analizarlas** primero, diseccionando las partes en las que están estructuradas y reconociendo, de cada una de esas partes, sus capacidades de automatización y sus principales retos. Así, en una etapa posterior, se podrán aprovechar esas cualidades de la mejor forma posible para sintetizar aplicaciones que automaticen el Web de forma más eficiente y menos costosa.

Acciones básicas

A pesar de la clara e intrínseca heterogeneidad del Web, mencionada en el apartado 1.4 y manifiesta, no sólo en sus datos, sino también en sus formas de presentación, lo cierto es que, por debajo de toda esta visible heterogeneidad, los principios bajo los que se sustentan las tareas en el Web son sencillos. De hecho, la simplicidad de manejo de cualquier browser no es más que un fiel reflejo de que, efectivamente, las partes de una tarea realizable

con una navegación en el Web se reducen a un conjunto limitado y sencillo de **acciones básicas**.

El conjunto de posibles tareas que se pueden realizar en el Web es inmenso. De hecho, no para de crecer en tanto en cuanto es cada vez mayor el número de aplicaciones que se encuentran disponibles a través del Web. Sin embargo, los innumerables servicios del Web, sean éstos simples páginas o bien complejas aplicaciones manejadas por varios formularios, pese a sus aparentes diferencias visuales, tienen en común el hecho de estar todos ellos fundamentados en alguna secuencia (o secuencias) de un reducido conjunto de acciones básicas que resultan ser **comunes a todas las tareas.** Cada tarea realizable en el Web se puede desarrollar efectuando una secuencia particular de este conjunto de acciones básicas. Por ejemplo, una típica tarea de chequeo de cuentas de correo electrónico en un portal de correo Web puede implicar una acción básica de solicitud de un documento donde aparece un formulario de identificación que, una vez rellenado y enviado al servidor, permite el acceso a una bandeja de correo entrante en la que aparece una tabla con los principales atributos de los mensajes almacenados, ordenados por fechas y en los que es posible distinguir entre mensajes leídos y no leídos. Cada uno de esos mensajes a su vez puede ser leído si se sigue convenientemente un enlace. También puede ser borrado, si se chequea su opción de selección correspondiente y después se pulsa al botón de borrado. Otras acciones, como responder o reenviar también son posibles. Secuencias distintas de este reducido conjunto de acciones básicas constituyen el quehacer diario del tráfico Web actual al que dan servicio los numerosos servidores Web conectados a la Red. Para la realización de una tarea Web, el usuario debe realizar la ejecución en secuencia ordenada, de cada una de las acciones básicas que conforman esa tarea.

Tipos de acciones básicas

Las acciones básicas que forman parte de una tarea no son todas de la misma naturaleza, pudiendo distinguirse entre acciones básicas implícitas y acciones básicas explícitas.

Las acciones básicas *implícitas* son aquellas que no deben trascender al usuario y que son realizadas automáticamente por el browser en una navegación manual. En ellas, el usuario no es necesariamente emprendedor de la acción. Se trata de acciones que se encuentran gestionadas internamente por los browsers, sin que el usuario deba proporcionar instrucciones explícitamente para ello. Ejemplos de acciones básicas implícitas son todas las referentes

a la gestión de cabeceras HTTP (gestión de cookies, redireccionamientos, identificación de usuario y de agente de usuario, preferencias en los formatos aceptados o en el idioma, ...), o las que se deben realizar necesariamente con el contenido del cuerpo de la respuesta HTTP (corrección interna de errores de estructura en el documento, seguimiento implícito de enlaces, acciones embebidas en lenguajes de scripting, ...).

Las acciones básicas **explícitas** son el conjunto de acciones que, en una navegación manual, deben trascender al usuario, ya que el browser no puede o no debe emprenderlas por su cuenta. Es por ello que el browser debe esperar instrucciones del usuario, típicamente de forma interactiva, para poder continuar. De esta forma, el usuario se acaba convirtiendo necesariamente en el emprendedor de la acción. Ejemplos de acciones básicas explícitas son el seguimiento explícito de enlaces, el rellenado y envío de formularios, la extracción de datos relevantes del documento o el procesamiento que debe realizarse con los datos recolectados.

Como puede apreciarse, las acciones básicas explícitas tienen un nivel de abstracción más elevado y cercano al usuario, mientras que las acciones básicas implícitas tienen un nivel de abstracción más bajo y más cercano a los aspectos técnicos de los protocolos HTTP y de los formatos de publicación en el Web. Como puede comprobarse en la tabla 2.1, que refleja las principales diferencias entre las acciones básicas explícitas e implícitas, las primeras afectan constantemente al usuario de la navegación manual. Por el contrario, esas acciones son atendidas por un programa que sustituye al usuario en las aplicaciones de navegación automatizada, tanto genérica como particularizada. Las acciones implícitas, por su parte, diseñadas para ser resueltas sin conocimiento explícito del usuario, son transparentemente gestionadas por los browsers en la navegación manual. En el caso de la navegación automatizada, lo deseable sería que la plataforma de navegación (el programa que navegue basándose en metadatos, o las bibliotecas usadas para la construcción del programa de navegación particularizada) ejecutaran todas esas acciones con la misma transparencia de un browser. No obstante, la falta de buenas plataformas de navegación automatizada para el Web con soporte para todas estas posibles acciones muchas veces acaba obligando al programador a introducir su propio código de tratamiento para suplir las carencias de las plataformas de navegación. Ello suele ser una fuente de encarecimiento de costes de estos programas, según el nivel de soporte que de estas acciones tenga la plataforma.

	Acción básica explícita	Acción básica implícita
Navegación manual	Usuario	Browser
Navegación automática genérica no adaptada	Robot	Según programa
Navegación automática genérica adaptada	Metadatos	Agente Web Semántico
Navegación automática particularizada	Programador	Bibliotecas+programador
Nivel de abstracción	Cercano a la tarea	Cercano a formatos y protocolos
Ejemplo	Enviar un formulario relleno	Gestionar una cookie

Cuadro 2.1: Diferencias entre acciones básicas explícitas e implícitas

2.1. Acciones básicas implícitas

Las acciones básicas implícitas son realizadas por muchos browsers sin que sus usuarios sean muchas veces conscientes de ello. Pese a su relativamente bajo conocimiento por muchos usuarios, las acciones básicas implícitas son necesarias para aspectos tan fundamentales como el mantenimiento de sesiones HTTP, las restricciones de acceso a contenidos, la adecuación del Web a preferencias del usuario, o la ejecución de múltiples comportamientos internos necesarios para la correcta navegación por cualquier página accesible desde el Web. A continuación aparecen detalladas algunas de las acciones básicas implícitas más importantes.

2.1.1. Gestión de cabeceras HTTP

El protocolo HTTP establece la posibilidad de intercambiar en varios campos situados en sus cabeceras, información relativa a las peticiones y respuestas intercambiadas. Estas cabeceras pueden ser usadas tanto por el cliente como por el servidor HTTP para obtener información que necesiten del otro extremo con el fin de mantener un diálogo correcto. La mayoría de esas cabeceras permanecen inalterables a lo largo de ese diálogo, pero otras muchas van cambiando a lo largo de ese diálogo, por lo que deben ser adecuadamente gestionadas para poder realizar correctamente la petición HTTP.

En la tabla 2.2 aparecen las principales cabeceras que **deben gestionar los clientes** del protocolo HTTP. Cada una de esas cabeceras tiene su especial importancia. Por ejemplo, algunos servidores Web presentan páginas de error cuando son accedidos por programas que no acreditan ser *Microsoft Internet Explorer* en la cabecera de identificación del cliente, pese a que sus contenidos sean en realidad navegables por otros browsers. En otros casos, no

especificar correctamente una cookie o el campo Referer en una petición a un servidor puede suponer la pérdida de la sesión con el servidor. No indicar que se acepta HTML como formato, puede implicar la suposición por parte del servidor de que debe responder con documentos en formato WML o quizá en texto plano. No indicar que el idioma preferible es el español puede implicar que las páginas vengan por defecto en inglés.

Nombre de cabecera	Emitida por	Significado	Ejemplo
User-Agent	Cliente	Identificación del cliente	Mozilla/4.6
Accept	Cliente	Formatos aceptados	text/html, image/*
Accept-Language	Cliente	Idiomas preferibles	en, es-ES, es
Referer	Cliente	URL desde la que se sigue el enlace	http://www.yahoo.es/
Cookie	Cliente	Valor almacenado en el cliente	NOMBRE=VALOR
Authorization	Cliente	Identificación para acceso restringido	Base 64
Set-Cookie	Servidor	Valor almacenable en el cliente	NOMBRE=VALOR
WWW-Authenticate	Servidor	Acceso restringido	"AccessRestreint"
Content-Type	Cliente o Servidor	Formato del documento o petición	text/html
Content-Length	Cliente o Servidor	Tamaño del documento o petición	12354

Cuadro 2.2: Principales cabeceras gestionadas por los clientes del protocolo HTTP

2.1.2. Gestión de errores en la comunicación con el servidor

Las comunicaciones con el protocolo HTTP pueden fallar si existe una sobrecarga en la conexión TCP/IP, el servidor al que se intenta conectar está inaccesible o su respuesta tarda un tiempo considerado demasiado alto para lo razonable por la tarea. Dependiendo de la gravedad del fallo (no es lo mismo que haya un fallo de transferencia en una página HTML que lo haya en la transferencia de una de sus imágenes), el error deberá hacerse constar con mayor o nivel de severidad al usuario o al programa que dirija la tarea para que tome las medidas que considere oportunas.

2.1.3. Reparación interna de páginas mal construidas

Una vez obtenida una página del Web y solventados los problemas de comunicación con el servidor, debe procederse a analizar la respuesta, nor-

malmente contenida en una página HTML. Lamentablemente, tal y como se mencionaba en el apartado 1.4.3, muchas de las páginas que se encuentran en el Web no cumplen el conjunto de normas básicas de construcción. Por esa razón, resulta necesario reparar internamente esos errores con el fin de permitir el análisis de esa respuesta por el cliente, ocultando al usuario esos errores en la medida de lo posible. Este trabajo *extra* en el lado del cliente debe ser realizado debido a la tradicional permisividad ante errores en las páginas obtenidas de los servidores.

2.1.4. Seguimiento implícito de enlaces

Una vez correctamente analizada una página obtenida del Web, es necesario identificar todos aquellos elementos multimedia que forman parte de la misma con el fin de seguir los enlaces que implícitamente deban visitarse. El seguimiento de enlaces puede ser entendido como **implícito**, si es el browser el que decide recuperar ese documento, quizá porque forme parte integrante del que acaba de ser descargado y analizado, o **explícito**, si es el usuario el responsable de la activación del enlace. Por ejemplo, los enlaces que en HTML se definen con las etiquetas `a` o `area` son normalmente seguidos de forma explícita, porque no deben ser visitados a no ser que el usuario explícitamente lo solicite. Por el contrario, en otros tipos de enlaces que en HTML se definen con etiquetas como `frame`, `img`, `script`, `link` u `object`, es el browser quien determina si deben ser seguidos de forma explícita o implícita. Si, por ejemplo, se desea emular a un browser gráfico como Mozilla [12] o Microsoft Explorer [91], los seguimientos de los enlaces definidos en las etiquetas anteriores, salvo los *roles* explícitos de `link` (*next*, *previous*, *table of contents*, ...), son implícitos, pues se entiende que un browser gráfico normalmente descarga implícitamente de cualquier página del Web todos sus componentes, como son los marcos, imágenes, scripts externos y hojas de estilo externas. Por el contrario, cuando se desea emular a un browser textual como Lynx [10], el seguimiento de los anteriores enlaces se vuelve, si no explícito, muchas veces incluso inaccesible, porque se entiende que hay documentos enlazados que no son adecuadamente procesados por el browser. En la tabla 2.3 puede verse un esquema resumido del comportamiento de estos enlaces dependiendo del browser en el que se utilizan.

Browser	a, area	img, frame	link	script, object
Gráfico	Explícito	Implícito	Implícito	Implícito
Textual	Explícito	Explícito	Limitado	Inaccesible
Braille	Explícito	Limitado	Limitado	Inaccesible

Cuadro 2.3: Tipo de seguimiento de enlaces HTML dependiendo del browser

2.1.5. Ejecución de comportamientos embebidos en las páginas

Una vez han sido correctamente descargados todos los elementos multimedia de las páginas, los elementos dinámicos de las mismas (rutinas JavaScript o JScript, Applets, controles ActiveX, animaciones en Flash, otros plugins, ...) pueden empezar a funcionar. Se trata de las rutinas que, mediante pequeños programas insertados en las páginas Web y, gracias a un intérprete, máquina virtual o plugin embebido en el browser, permiten al diseñador de la página tener acceso a ciertos recursos del browser del usuario. Las rutinas JavaScript, que suelen ser las más usadas, permiten controlar un amplio espectro de acciones típicamente manejadas por el browser. Por ejemplo, son capaces de manipular aspectos tan diversos como las propiedades de visualización de los elementos de las páginas (tamaños, posiciones, colores, visibilidad, ...) , la descarga condicionada de elementos multimedia de una página, manipular la base de datos de cookies o modificar al gusto del diseñador de la página la semántica del comportamiento de los controles de los formularios. Tanto es así, que muchas páginas resultan innavegables si el browser con el que se las intenta acceder no dispone de un intérprete ajustado a las especificaciones del JavaScript utilizado en la página.

Si la plataforma de navegación no dispone de estos intérpretes o estos plugins, los comportamientos embebidos no son ejecutables. Ello puede no suponer un problema serio para el procesamiento de páginas con comportamientos embebidos que sólo afecten a aspectos de visualización, siempre que existan contenidos alternativos ofrecibles al usuario o, simplemente no aporten datos relevantes. Sin embargo, puede suponer serios problemas para la accesibilidad cuando esos comportamientos tienen un papel en la navegación.

Este tipo de rutinas tiene una característica muy importante desde el punto de vista de su automatización. Al tratarse de comportamientos que vienen pre-programados en elementos multimedia de las páginas, su comportamiento, no se encuentra por lo tanto pre-programado en ningún browser, es decir, no es conocible a priori, por lo que para su correcto funcionamiento se necesita del correspondiente soporte para la ejecución. Sin embargo, dis-

poner de ese soporte puede ser difícil o costoso para muchos usuarios que no siempre lo encuentran disponible en browsers o plataformas de navegación automatizada.

2.1.6. Soporte para otros protocolos

En ocasiones, se deben enviar mensajes por correo electrónico. En otras ocasiones, algunos documentos deben recuperarse usando otros protocolos como FTP. También es posible que sea necesario acceder a ciertas páginas mediante protocolos seguros, principalmente SSL. Forma parte de las acciones implícitas de la plataforma de navegación saber gestionar adecuadamente estos protocolos, de forma que el usuario no sea consciente de esos detalles de bajo nivel de cada uno de ellos.

2.1.7. Tratamiento adecuado de cada campo de formularios según su forma de rellenado

A la hora de asociar valores a los campos de los formularios, es necesario tener en cuenta las reglas de rellenado de cada campo del formulario reflejadas en la tabla 2.4. Por una parte, debe considerarse el **tipo de valor** que indica la naturaleza del valor asociado a cada campo del formulario. Principalmente, pueden distinguirse textos libres especificados por el usuario, ficheros que deben ser enviados del cliente al servidor y valores ya especificados en la página por el servidor. Por otra parte, debe considerarse la especificación de rellenado por el usuario, es decir, aquello que el usuario debe proporcionar para que dicho campo de formulario quede adecuadamente relleno. Principalmente, puede distinguirse entre el rellenado directo, que consiste en el que el usuario proporciona directamente el valor con el que desea rellenar el formulario, o bien un rellenado indirecto, en el que el usuario especifica un criterio de selección que sirve para decidir las opciones que deben quedar marcadas y las que no. El criterio de selección consiste normalmente en un criterio booleano que, aplicado a cada una de las opciones seleccionables, indica si esa opción debe admitirse o no como seleccionada. Los criterios de selección pueden ser de dos tipos, sencillos o múltiples, dependiendo de si, cuando son aplicados a un conjunto de varias opciones, deben indicar una o varias de esas opciones como seleccionadas. Este *acomodamiento*, o conjunto de restricciones establecidas sobre el conjunto de valores posibles con los que se puede rellenar un formulario, se encuentra ya facilitado por los browsers y es una de las acciones básicas implícitas que influyen en la forma de rellenar

cualquier formulario.

Campo	Tipo de valor	Rellenado por usuario
textarea	Texto libre	Rellenado directo
input.text, input.password	Texto libre sin saltos de línea	Rellenado directo
select, input.radio	Valor especificado por el servidor	Criterio de selección
select.multiple, input.checkbox	Valores especificados por el servidor	Criterio de selección (múltiple)
input.file	Fichero local	Path al fichero
input.hidden	Valor especificado por el servidor	Oculto al usuario
input.button	Llamada a JavaScript	No
input.reset	Reinicio del formulario	No
input.submit, input.image	Envío	Criterio de selección

Cuadro 2.4: Tipo de rellenado de campos de formularios HTML

2.1.8. Creación de query-string a partir de un formulario relleno

Antes de enviar a un servidor un formulario relleno, es necesario codificar la información que encierra para su envío, de forma que sea correctamente procesable en el servidor. Esta información codificada, denominada query-string, es enviada como parte de la correspondiente petición HTTP al servidor. En el caso de que la petición sea del tipo POST, el query-string debe formar parte del cuerpo de la petición, por lo que irá aislado aparte de las cabeceras HTTP. Por el contrario, si el formulario se envía con un comando GET, el query-string es concatenado al final de la URL que se desea visitar.

La acción de codificación de formularios rellenos debe tener en cuenta su estructura, respetando el orden y el tipo de cada uno de los campos que componen el formulario. A la hora de crear el query-string, se deben establecer parejas campo-valor, independientemente de cómo se hubieran rellenado los campos de ese formulario. Por otra parte, en la codificación del query-string puede frecuentemente jugar también un papel importante el JavaScript. Muchos de los campos de los formularios dedicados a controlar eventos JavaScript no deben ser enviados al servidor. Los botones de envío no pulsados tampoco deben ser enviados al servidor.

2.2. Acciones básicas explícitas

Las acciones básicas explícitas son las que definen los pasos en los que está involucrado el usuario durante la ejecución de una tarea en el Web. Realizarlas con un navegador suele ser un buen punto de partida para empezar a construir el esqueleto básico de una aplicación de navegación automatizada. Las tareas del Web son, por lo tanto, especificadas basándose en este tipo de acciones. Algunas de ellas, sin embargo, no son siempre necesarias en todos los pasos. La tabla 2.5, muestra cada una de esas acciones en una fila, para comparar su capacidad de automatización desde los puntos de vista de la navegación manual y de la navegación automática. Como puede apreciarse, desde el punto de vista de la navegación manual, todas ellas, en mayor o menor medida, con mayor o menor soporte por parte del browser, son responsabilidad del usuario. Desde el punto de vista de la navegación automatizada, estas acciones deben recaer en un programa, por lo que se mencionan las partes de ese programa que están afectadas por cada una de esas acciones. Por ejemplo, las acciones de seguimiento de enlaces y envío de formularios apenas suelen suponer la correspondiente llamada a las primitivas GET o POST del protocolo HTTP, a la que sólo hay que parametrizar convenientemente. Sin embargo, el resto de las acciones básicas explícitas no tiene un coste tan *reducido* como ése habitualmente, ya que, al ser dependientes de la tarea, deben ser programadas con código de usuario.

Acción básica explícita	Navegación manual	Navegación automatizada
Extracción de datos relevantes	Usuario	Reglas de extracción (regularidad estructural)
Estructuración de datos semiestructurados	Usuario	Repositorios estructurados
Seguimiento explícito de enlaces	Usuario + Browser	Llamada a primitiva GET
Rellenado de formularios	Usuario + Browser	Metadatos, código de programador
Envío de formularios	Usuario + Browser	Llamada a primitivas POST/GET
Procesamiento de datos	Usuario	Rutinas de usuario, programas externos, ...

Cuadro 2.5: Acciones básicas explícitas

2.2.1. Extracción de datos relevantes

La extracción de datos relevantes implica la selección de los datos considerados relevantes embebidos en las páginas Web y su extracción para posteriores procesamientos. Estos datos, cuya relevancia se encuentra descrita en

el apartado 1.4.4, se suelen encontrar repartidos por varias páginas o marcos. Cuando no se trata de ficheros multimedia, lo normal es que se encuentren en forma de simple información textual (e.g.: precios, titulares, mensajes, teléfonos, direcciones, cotizaciones, fechas, cantidades, ...) típicamente embebida en páginas HTML junto con otra mucha información que puede no ser relevante para la tarea (publicidad, marcado estructural orientado a la visualización, datos relevantes para otras tareas, ...).

La capacidad de extraer datos del Web, es sin duda de las más importantes, pues juega un papel importante en todos los pasos ejecutados entre la navegación por el Web. Seguir un enlace implica seleccionar en primer lugar el enlace que debe ser seguido y extraer de él la dirección a la que apunta. Enviar un formulario implica seleccionar en primer lugar todos los campos que forman parte del mismo y rellenar cada uno conforme a su naturaleza y a los objetivos del usuario, para después activar la acción del formulario. Otros procesamientos más complejos definidos por el usuario, como comparaciones, integración de datos y otros comportamientos, necesitan también la realización de complejas extracciones de datos. De esta forma se prescinde de todos aquellos datos que aparecen en las páginas pero que no intervienen en la tarea.

En el caso de la navegación manual basada en browsers, el usuario debe visualizar normalmente pantallas enteras para poder **detectar visualmente** la información que le interesa. Es normal que para ello deba examinar varias pantallas y ventanas o hacer *scrolling*. Desafortunadamente, los browsers de hoy en día no reciben especificaciones acerca de cuáles son los datos relevantes para los usuarios y cuáles pueden ser ignorados, con el fin de destacar los primeros y ocultar los últimos (esta labor quizá podría ser emprendida con scripts definibles por los usuarios aplicados a documentos del Web, pero ello no es una solución siempre factible). La única función de un browser es mostrar un documento de la mejor forma posible y ejecutar sus órdenes interactivas, siendo imposible destacar para cada usuario los datos que son interesantes para él y su tarea. Las opciones de destacado de partes de documentos están a merced de los autores de los documentos, no de sus lectores, algo que sin embargo algunos trabajos como [48] sí han logrado hacer basándose en una personalización que no todos los sitios Web ofrecen. Aunque la simple lectura de textos sobre las pantallas de un ordenador puede ser suficiente para algunos usuarios a la hora de pasar a realizar su siguiente acción, lo cierto es que el Web se caracteriza por presentar **demasiada información** que presenta costes prohibitivos para ser procesada por personas, razón por la cual una separación automatizada de los datos relevantes respecto del resto de datos no relevantes resulta conveniente en esos casos.

En el caso de la navegación automatizada, la extracción de datos no está basada en la visualización, sino que consiste en una selección de datos relevantes dentro de documentos semiestructurados, lo cual no es siempre sencillo y es además claramente dependiente de la estructura interna de esos documentos. Para ello, pueden usarse **reglas de extracción** de datos basadas en una **regularidad estructural** seleccionando aquellos datos que cumplan un formato esperado. Sin embargo, esta estructura de marcado en la que están basadas estas reglas está normalmente muy orientada a los aspectos de visualización, por lo que las reglas de extracción de datos, aparte de ser de las partes con mayor presencia en los programas, son también de las más frágiles, ya que cualquier cambio en la estructura esperada de las páginas afecta directamente a esas reglas. La fácil construcción de estas reglas de extracción de datos es vital para conseguir un bajo coste, no sólo de desarrollo, sino también de mantenimiento.

En el caso de la navegación manual, esta labor recae completamente en el usuario. El browser prácticamente no interviene ni realiza otra labor más que presentar los datos en la pantalla junto con el resto de la información, sin tener capacidad para tan siquiera resaltar el dato para el usuario, pues no tiene forma de saber cuál de los datos que figuran en la página es el dato que interesa al usuario para su tarea. En el caso de la navegación automática, al tratarse de un tratamiento específico y dependiente de la estructura de las páginas, de los datos, y de la tarea que los va a utilizar, esta acción ***no se encuentra pre-construida*** *en una biblioteca genérica de la plataforma, por lo que debe ser programada en reglas de extracción definibles por el usuario.*

2.2.2. Estructuración de datos semiestructurados

Los datos extraídos pueden ser necesarios más de una vez a lo largo de la ejecución de una tarea, por lo que conviene que sean convenientemente almacenados en un repositorio estructurado. Desde el punto de vista de la navegación manual, esta labor recae en la responsabilidad del usuario, quien habitualmente suele resolver el problema memorizando el dato recientemente visualizado (típicamente en su memoria a corto plazo), apuntarlo en un papel o, en el mejor de los casos, recurrir el conocido uso del *copiar y pegar* en la ventana de otra aplicación. Sin embargo, ni la mente humana, ni el papel, ni una ventana de un editor de texto son repositorios *adecuados* para el procesamiento automatizado desde un programa de ordenador. Además, los datos obtenibles del Web pueden ser muy voluminosos (de ahí el desbordamiento habitual que sufren la mayoría de los usuarios que navegan con browsers), por

lo que para poder almacenar convenientemente esos datos se necesitan **repositorios** capaces de almacenar grandes volúmenes de los mismos, muchas veces sin que su tamaño sea a priori limitable. Por esa razón, los programas de navegación automática suelen usar, no sólo variables de memoria, sino estructuras de datos de tamaño variable, como vectores, listas o ficheros, en los que ir almacenando los valores extraídos de las páginas para que puedan ser posteriormente procesados. Cuando se almacenan las partes seleccionadas de estos documentos en repositorios especializados, los datos seleccionados pueden ser convertidos a tipos de datos más fácilmente procesables habituales en los lenguajes de programación, como números, booleanos, fechas, cadenas de caracteres o, quizás, nodos de un árbol de documento.

En el caso de la navegación manual basada en browsers, el usuario es el responsable de almacenar los datos y por ello es él quien decide cómo almacenarlos. Normalmente los suele intentar memorizar o dejar en alguna ventana abierta del navegador que posteriormente esté accesible para poderla volver a leer, pero ello implica la necesidad de tener que volver a extraer de ella nuevamente los datos cada vez que se deseen manipular, y la posibilidad de perderlos si se siguen enlaces en la misma ventana. En el mejor de los casos, pueden almacenarse en otras herramientas externas, pero ello implica una ardua labor de estructuración con el fin de poder manipular eficientemente grandes volúmenes de información. En el caso de la navegación automática, al tratarse de un tratamiento específico y dependiente de la estructura de las páginas, de los datos, y de la tarea que lo va a utilizar, esta acción ***no se encuentra pre-construida*** *en una biblioteca genérica de la plataforma, por lo que debe ser programada en las correspondientes sentencias de almacenamiento en repositorios estructurados.*

2.2.3. Seguimiento explícito de enlaces

Los datos en el Web se suelen encontrar distribuidos en múltiples documentos que, por lo tanto, deben ser recuperados. Para ello suele usarse el protocolo HTTP, haciendo un seguimiento explícito de enlaces conforme a la tabla 2.3. A veces las direcciones de esos documentos no son conocidas a priori por el usuario, sino que tienen que ser **obtenidas** dinámicamente desde otras páginas ejercitando la navegación. En los casos más sencillos, bastará con visitar una dirección Web en la que se sabe que figuran los datos que se desea consultar. En otros casos, es necesario establecer una sesión desde la página principal del sitio Web de forma que hay que seguir varios enlaces y rellenar varios formularios antes de acceder finalmente a la página

que contiene el dato.

En el caso de la navegación manual, el seguimiento explícito de enlaces consiste en una labor semiautomática, asistida por el browser, donde la labor del usuario se reduce a seleccionar y activar los enlaces que le interesan. En el caso de la navegación automática, esta acción ***se encuentra ya completamente pre-construida*** *en las primitivas de varias bibliotecas utilizables desde distintos lenguajes de programación para lanzar comandos GET, por lo que típicamente la labor de programación se reduce a parametrizar la llamada a esta primitiva.*

2.2.4. Rellenado de formularios

El rellenado de un formulario Web consiste simplemente en asociar uno o varios valores a cada uno de sus campos (también es posible dejar algunos de ellos vacíos). Los valores con los que se rellenan esos campos (tal y como viene expresado en la tabla 2.4) pueden venir definidos por el usuario, o pueden venir predefinidos en el propio formulario, pero en cualquier caso es el usuario el responsable de establecer aquellos valores que le interesen para su tarea. Por otra parte, puede haber campos de formularios que no son manipulados (porque no los ha querido rellenar) o que no son manipulables (porque están ocultos al usuario), en cuyo caso, conservan el valor con el que vinieron rellenados por defecto en el formulario.

En el caso de la navegación manual, se trata de una labor semiautomática asistida por el browser, donde la labor del usuario se reduce a interactuar con los campos de los formularios. En el caso de la navegación automática, al tratarse de un tratamiento específico y dependiente de la estructura del formulario, y de la tarea que lo desee rellenar, esta acción ***no se encuentra pre-construida*** *en una biblioteca genérica de la plataforma, por lo que debe ser programada en código definido por el usuario. Normalmente la complejidad del rellenado de cada campo es dependiente de la estructura interna de representación de ese formulario, pues es normalmente sobre ella donde se realizan estas modificaciones para después proceder a la orden de envío del formulario, según la siguiente acción explícita. En el mejor de los casos, con una buena representación donde exista una lista de campos recorrible y unas primitivas de modificación acordes con la semántica de cada campo, esa labor puede realizarse en una simple línea de código para cada uno de esos campos.*

2.2.5. Envío de formularios

Una vez que un formulario se encuentra ya relleno, la información recogida en él puede ser enviada al servidor para que sea procesada.

En el caso de la navegación manual, se trata de una labor semiautomática, asistida por el browser, donde la labor del usuario se reduce a seleccionar un botón de envío (en el caso en el que haya varios) y pulsarlo. En el caso de la navegación automática, esta acción ***se encuentra ya completamente pre-construida*** *por las primitivas de varias bibliotecas utilizables desde distintos lenguajes de programación para lanzar comandos GET o POST, por lo que típicamente apenas se necesita programar y parametrizar la llamada a esta primitiva.*

2.2.6. Procesamiento de datos

Una vez que los datos del Web han sido recuperados, y homogeneizados a un formato estructurado, su manipulación no dista de la que puede haber en cualquier tarea de tratamiento de datos (no necesariamente involucrada en el Web). Operaciones tales como comparaciones, acumulaciones, reordenaciones, operaciones aritméticas o lógicas o de procesamiento de textos pueden ser combinadas según las necesidades específicas de manipulación de información que requiera la tarea.

En el caso de la navegación manual, el procesamiento que típicamente realizan los browsers a las páginas que recuperan del Web se limita a la mera visualización en las pantallas de los ordenadores, facilitando, eso sí, el control de todos los aspectos visuales y permitiendo al usuario la posibilidad de solicitar nuevos documentos del Web mediante el resaltado en la visualización de zonas activables del documento por el usuario mediante teclado o ratón para el seguimiento de enlaces. Cualquier otro tipo de procesamiento queda delegado en el usuario.

Muchas veces, ciertamente, estos tratamientos de datos en el Web son comparaciones sencillas o labores que manejan poca información, pero, cuando el volumen de datos es algo elevado, o cuando el tratamiento que se pretende realizar con esos datos empieza a incluir operaciones aritmético-lógicas un poco más complicadas que las simples comparaciones detectables a simple vista y que escapan del fácil cálculo mental, el procesamiento manual de esos datos se convierte en una labor realmente tediosa.

En el caso de la navegación automática pueden definirse rutinas que pro-

cesen esos datos de otra forma más adecuada para las tareas. Mediante la programación de rutinas definibles por el usuario, que reciban por argumento los datos obtenibles durante la navegación, el tratamiento de estos datos puede ser realizado por el ordenador. Simplemente deberán procesarse los datos relevantes que se encuentran almacenados en repositorios programables, conforme a los objetivos establecidos en la tarea. También es posible que esos procesamientos puedan estar ya implementados en alguna herramienta externa. En esos casos, el tratamiento consistirá en invocar a esa herramienta como un proceso externo, enviarle los datos para que los procese y esperar de ella los resultados. En el mejor de los casos, el procesamiento puede ser tan sencillo como la simple impresión por pantalla de unos simples datos obtenidos, para lo cual es posible programar esos comportamientos con sentencias sencillas dentro del mismo programa principal. La elección sobre dónde programar este tipo de comportamientos dependerá de la complejidad de los mismos, del hecho de que ya pudieran estar programados en algún programa legado y de la capacidad del programador para reutilizar ese código.

En el caso de la navegación manual basada en browsers, el usuario es el responsable de realizar este procesamiento de datos, normalmente de forma mental y sin soporte por parte del browser. En el caso de la navegación automática, al tratarse de un tratamiento específico y dependiente de los datos, y de la tarea que lo va a utilizar, esta acción ***no se encuentra pre-construida*** *en una biblioteca genérica de la plataforma, por lo que debe ser programada en rutinas definidas por los usuarios, que no necesariamente serán dependientes de la cambiante estructura de las páginas al haber sido los datos convenientemente estructurados en una fase anterior.*

2.3. Subsanación de las faltas de soporte de la plataforma de navegación

Finalmente, tal y como se verá en el apartado 3, no todas las plataformas tienen un soporte completo a la ejecución de aspectos de navegación. Por ejemplo, muchas plataformas carecen de soporte a la interpretación de rutinas JavaScript, que se encuentran habitualmente embebidas en las páginas visitadas. En estas plataformas no está soportada, por lo tanto, la navegación basada en JavaScript, en la que las URL que deben visitarse no figuran explícitas en los enlaces que se pretende seguir, sino que dichas direcciones deben computarse por alguna rutina JavaScript activable por algún evento provocado por el usuario con el ratón o el teclado. Para poder navegar en

estos sitios desde este tipo de plataformas, el programador debe subsanar con sus propias líneas de código las acciones que simulen el comportamiento de esas rutinas JavaScript. En otras plataformas no existe un conveniente soporte de las cabeceras HTTP o una adecuada creación del conveniente query-string a partir de cada formulario relleno, por lo que el usuario debe programar estos comportamientos implícitos con su propio código. Todas aquellas acciones que, no teniendo soporte en la plataforma, sean significativas para la navegación, deben ser suplidas con código definido por el usuario. Dicho código suele tener un coste significativamente elevado. Por esta razón, y para minimizar este coste, es importante, por lo tanto, escoger una buena plataforma de navegación.

Soporte de los browsers a las acciones básicas explícitas

Tal y como puede verse en la tabla 2.5, en la navegación manual, el browser da un soporte semiautomático a tres de las acciones más sencillas (seguimiento explícito de enlaces y rellenado y envío de formularios), dejando al usuario la responsabilidad de realizar las otras acciones sin ningún tipo de asistencia por parte del browser. Desde la extracción de datos relevantes hasta el procesamiento que pueda necesitar realizarse sobre esos datos, el usuario que utiliza browsers es quien debe encargarse de todo.

Capítulo 3

Estado de la cuestión

En este capítulo se realiza un repaso de los principales conceptos y técnicas desarrolladas hasta el momento en el ámbito de la integración de datos semiestructurados, así como de otras técnicas, no ya de automatización de tareas en el Web, sino, más genéricamente, de navegación automática en el Web.

3.1. Consideraciones previas

Antes de comentar esos trabajos aplicados al tema específico de la automatización de tareas en el Web, conviene mostrar cómo algunos sitios Web afrontan actualmente el uso de sus aplicaciones.

- Algunos sitios Web, muy pocos en términos relativos, proporcionan aplicaciones ejecutables alternativas a los browsers para proporcionalidad una mayor facilidad de manejo a aquellos usuarios que realizan un número elevado de transacciones. En estos casos, lo habitual suele ser que los desarrolladores de la aplicación proporcionen al usuario una forma de acceso indirecto a la aplicación con la que interactúan de forma que el interfaz no esté basado en el browser, sino que en un programa ejecutable capaz de automatizar varias de estas transacciones minimizando parcial, pero sensiblemente, la interactividad solicitada al usuario. Por ejemplo, eBay [6] o el antiguo QXL (recientemente fusionado con Aucland [1]) han proporcionado a sus mejores vendedores una aplicación ejecutable (para uso exclusivo en entornos Windows) que permite la publicación de múltiples subastas en la red con un sólo click

de ratón. Por otro lado, algunas operadoras de contratación de acciones en bolsa, como por ejemplo Consors [4] proporcionan a sus clientes una interfaz Java (normalmente un applet ejecutable en el navegador) para facilitar a sus usuarios más activos una plataforma de introducción de órdenes de compra-venta más manejable que el browser cuando se trata de un número elevado de operaciones o se requieren algunas funcionalidades como información en tiempo real. Si bien la elección de la plataforma tecnológica puede ser muy dispar (existen igualmente soluciones basadas en controles ActiveX y otras variantes), lo cierto es que este tipo de aplicaciones a veces no proporcionan una funcionalidad completa, ya que en ocasiones se limitan las opciones de la versión *interactiva* basada en HTML y en la comunicación a través del browser, por lo que fácilmente pueden no contemplar todas las funcionalidades que desean los usuarios.

- Por otro lado, muchas de las páginas que necesitan ser accedidas, no tienen una interfaz especialmente amigable cuando se les usa desde otro tipo de browser distinto a aquél para el que han sido diseñadas. Dicho de otra forma, sus páginas son poco accesibles.

En cualquier caso, pocos diseñadores de sitios Web están dispuestos a facilitar que sus páginas sean navegadas por programas automatizados (robots) en lugar de por personas usando browsers. Sin duda influyen más razones sociológicas (miedo a perder atracción en la publicidad, miedo a que los contenidos propios sean aprovechados por terceros para hacer negocio sin que el verdadero propietario reciba beneficios, ...) que técnicas (miedo a ver sobrecargada la capacidad de respuesta de los servidores).

3.2. Automatización de aplicaciones interactivas

Sin duda, estos problemas (el de la menor funcionalidad de las opciones no interactivas de las aplicaciones y el de la baja amigabilidad del interfaz de algunas aplicaciones) afectan al ámbito de la automatización, pero no sólo a la del Web, sino, en general, al de cualquier aplicación diseñada para ser usada de forma interactiva. Dado que la problemática de la automatización de aplicaciones interactivas [149] es anterior al nacimiento del mismo Web, conviene sin duda analizar algunas de sus conclusiones más relevantes para aprovechar así la experiencia desarrollada.

3.2.1. Lenguaje Expect

Aunque existen múltiples trabajos enfocados en la automatización de aplicaciones interactivas en varios entornos, de todos ellos destaca sin duda `expect` [84]. En `expect`, mediante la utilización de un nuevo lenguaje de scripting específico similar al de un shell, se permite realizar fácilmente el control de programas interactivos lanzados en entornos Unix que estén preparados para leer del teclado de una terminal. A diferencia de una shell normal, `expect` resulta especialmente útil para emular al usuario desde el teclado cuando este tipo de fuente de datos no puede ser fácilmente redireccionado a un fichero para lectura o cuando es necesario responder adecuadamente a un *prompt* en un diálogo con la aplicación interactiva, con peticiones del programa y respuestas que deben introducirse por teclado en el momento en el que el programa interactivo las solicita.

Tal y como reza en ese trabajo, los tradicionales shells Unix tienen, sobre las aplicaciones que invocan, un control que se limita a la creación, espera y destrucción de procesos, así como las opciones con las que deben ser invocados al principio y el redireccionamiento a/desde ficheros pero no tienen apenas control sobre aquellos programas que necesitan interactividad durante su ejecución, dejando esa tarea relegada a que el usuario introduzca esos datos desde teclado. Mediante una filosofía de lanzamiento de ejecuciones similar a la de los shells, pero con extensiones para controlar la ejecución interactiva de estos programas, aplicaciones que hasta ese momento sólo podían usarse de forma interactiva, como `telnet`, `ftp`, `passwd`, `rlogin`, `crypt`, `fsck`, `sudo` o incluso otras para las que podía emular al usuario ante otro usuario, como por ejemplo `talk`, y en general, cualquier aplicación (incluyendo las que se pudiera construir el usuario por su cuenta) que pudiera ser usada de modo interactivo desde teclado, pueden ser controladas automáticamente desde un programa capaz de **entender** y **proporcionar** los datos que cada una de esas aplicaciones muestran y solicitan del usuario de forma interactiva. Para ello, se emula al usuario sustituyéndolo por la aplicación de un conjunto de reglas condicionales capaces de detectar los distintos casos de peticiones *esperables* por las aplicaciones interactivas y así asociar a cada regla un conjunto de acciones de respuesta.

El lenguaje desarrollado en ese trabajo, llamado `expect`, actualmente instalado en muchas distribuciones de sistemas Unix, está basado en la sintaxis de TCL [97] y, entre las conclusiones reflejadas en [84] destaca, para ilustrar su uso con un ejemplo, como el script de la figura 3.1, diseñado para controlar mediante el diálogo interactivo con la aplicación Unix `ftp` el acceso a un sitio ftp anónimo, sustituyó a un programa *equivalente* (que hacía lo mismo) es-

crito en lenguaje C, pero que tenía un tamaño aproximado de 60K. El script de la figura 3.1 espera a recibir la palabra `Name` del programa `ftp` antes de enviarle la palabra *anonymous* de la misma forma a la que espera a estar identificado correctamente antes de lanzar una petición de transferencia de ficheros.

```
#! /usr/bin/expect -f
spawn ftp [index $argv 1]
expect "*Name*"
send "anonymous\r"
expect "*Password:*"
send [exec whoami]
expect "*ok*ftp>*"
send "get [index $argv 2]\r"
expect "*ftp>*"
```

Figura 3.1: Script Expect para controlar ejecución interactiva de ftp

Quizá lo más llamativo de `expect` como lenguaje sea sin duda la gran diferencia de tamaño existente entre la solución escrita en él y la escrita en C, ambas para solucionar el mismo problema. La diferencia de tamaño se puede justificar en el hecho de que C es un lenguaje de programación imperativo de uso genérico, mientras que `expect` es un lenguaje de scripting de alto nivel de programación y orientado al mantenimiento del control del diálogo con cualquier aplicación y capaz de proporcionar distintas respuestas a la aplicación dependiendo de lo que ella muestre a su salida. `expect` es, por lo tanto, un claro ejemplo de lenguaje con un nivel de abstracción lo suficientemente elevado como para poder ser usado casi a nivel de especificación de requisitos.

Sin embargo, un lenguaje para la automatización, pese a que disponga de un alto nivel de abstracción, para poder ser aplicable a cualquier herramienta interactiva, debe ser capaz de controlar aspectos de bajo nivel. Para mostrar la flexibilidad del lenguaje `expect` en este sentido, el script de la figura 3.2 ilustra la capacidad de emulación de un usuario ficticio capaz de dialogar con otro (en un diálogo muy sencillo, pero que podría ser fácilmente adaptable) mediante el uso de la conocida herramienta `talk` de entornos Unix. El nivel de control de `talk` en este caso llega incluso a controlar aspectos como un modelo de tiempos variables entre pulsaciones de las teclas, dando al usuario que está al otro extremo de la comunicación la ilusión de que realmente está dialogando con una persona pese a que en realidad no deja de ser un programa que emula a un usuario. No ya en una aplicación como `talk` sino en cualquiera donde el número de posibles salidas pueda ser conocido, un conjunto completo de reglas y acciones que actúen en consecuencia pueden

automatizar completamente la gestión de una aplicación interactiva.

```
#! /usr/bin/expect -f
spawn talk usuario@dominio
set timeout 200
expect "*established*"
set send_human {.1 .3 1 .05 2}
send -h "This is only a test.. I swear \ Please don't bust me with expect"
expect "*\r*"
exec sleep 5
send -h "Ok, well see ya tomorrow . Bye\n"
exec sleep 3
```

Figura 3.2: Script Expect para controlar ejecución interactiva de talk

En resumen, expect aporta, para el manejo de aplicaciones interactivas, un **lenguaje de scripting** con las siguientes características:

- Alto nivel de abstracción
- Con orientación al diálogo con aplicaciones existentes
- Que reduce sustancialmente el tamaño y, por lo tanto el esfuerzo para crear y mantener, de aplicaciones capaces de automatizar a otras
- Capaz de automatizar prácticamente cualquier aplicación interactiva textual (no gráfica)
- Capaz de analizar la información que proporcionan las aplicaciones interactivas y estructurar su comportamiento basándose en esos casos
- Capaz de asociar acciones a cada uno de los casos que se espera que proporcione la aplicación interactiva
- Capaz de permitir al usuario gran flexibilidad para que defina sus propias reglas y sus propias acciones
- Capaz de emular al usuario controlando aspectos de bajo nivel
- Basado en la sintaxis de algún estándar conocido

Estas características han sido tenidas en cuenta para la automatización de tareas en el Web en el apartado 6.

3.3. Web Semántico

Uno de los grandes problemas del Web actual, desde el punto de vista de su procesamiento automatizado, es que está basado principalmente en HTML (formato difícil de entender por las máquinas al estar muy orientado a la mera visualización conforme al apartado 1.4.3). Teniendo esto en cuenta junto al hecho de que se espera que XML aún tarde varios años en implantarse como formato para los documentos en el Web, desde el W3C se ha estado promoviendo en los últimos años una ambiciosa iniciativa denominada el Web Semántico [32].

El objetivo del Web Semántico es el de permitir la navegación automatizada por parte de programas capaces de *entender* el significado de los datos que aparecen en las páginas del Web, gracias al hecho de que a éstas les acompañan unos **metadatos** (típicamente basados en RDF [123] u OWL [140]) capaces de asociar un significado a cada una de las partes del documento, describiendo con ontologías los datos que aparecen en las páginas Web. Las páginas así descritas con los correspondientes metadatos pueden ser asimilables a bases de datos procesables por cualquier tipo de programa. Así pues, un agente inteligente basado en motores de inferencia capaces de procesar los metadatos descriptivos de una página puede encontrar para el usuario la información que satisfaga sus requisitos de búsqueda con un mayor criterio que la simple búsqueda por palabras clave, cuando se busca en un conjunto de páginas unidas entre sí por hiperenlaces.

Siendo el Web Semántico una opción realmente prometedora para el futuro del Web y de la construcción dinámica de caminos de navegación, guiada por la consecución de objetivos, lo cierto es que todavía es una tecnología incipiente a la que aún le falta por demostrar sus capacidades en entornos complejos, habiendo sólo sido probada en entornos sencillos como [102]. En [64] se cuestiona cómo un Web dinámico puede tener asociadas páginas de metadatos estáticas, estableciendo diferencias entre las semánticas estáticas de estos ficheros declarativos creados aparte por personas y las semánticas dinámicas necesarias para manejar los datos del Web, que normalmente se dan en el contexto de un lenguaje de programación, abogando así por soluciones no precisamente declarativas. Por otro lado, las necesidades de los usuarios son mucho más complejas que aquellas a las que puede dar respuesta un buscador. Una vez delante de la página en la que debe empezar a trabajar, es necesario desarrollar una tarea cuyos pasos son normalmente conocidos por los usuarios y no es necesario que sean *deducidos*.

En lugar de definir programas envoltorio, comúnmente conocidos como

wrappers para cada fuente de datos accedida, el Web Semántico propone la utilización de programas de navegación genérica capaces de autoprogramar su navegación a cualquier sitio Web conforme a la información suministrada por los metadatos de ese sitio Web. Ello supone, en la práctica, delegar cualquier automatización de tareas en la construcción de una adecuada meta-información de un sitio Web, capaz de dirigir el comportamiento de estos programas de navegación genérica por ese Web de forma similar a la que lo navegaría un programa envoltorio.

El Web Semántico está en un estado aún inmaduro y carece del soporte necesario de muchas herramientas. Por otro lado, la mayoría de las páginas del Web también carecen de los correspondientes metadatos, y eso es algo que tardará tiempo en paliarse aun cuando el Web Semántico recale en la construcción de Webs. Por todo ello, es previsible que la utilización de las técnicas del Web Semántico tardarán aún bastante tiempo en poder explotarse masivamente en Internet para la automatización de tareas en el Web.

3.4. Mecanismos de construcción de programas de navegación automatizada

Los siguientes trabajos abordan de una u otra medida el tema de la automatización de la navegación en el Web.

En buena parte de los proyectos en los que se desarrollan trabajos para la especificación de aplicaciones para la Web, la gran mayoría de los esfuerzos se vuelcan en la especificación de aplicaciones accesibles desde el Web que sean funcionales y no produzcan errores a sus usuarios. Ejemplos de esta alternativa son XL [59] y Dicons [27] donde se definen lenguajes de especificación para la Web, pero en el lado del servidor, no en el lado de un cliente que desee automatizar el uso de esas mismas aplicaciones. Otros trabajos, como WebML [50] intentan aplicar el modelo Entidad-Relación de las bases de datos relacionales a ciertas páginas del Web. Tal modelado resulta enriquecedor en el sentido de que aporta una visión muy descriptiva y detallada del esquema de datos usado al publicar muchas de las páginas Web existentes hoy en día. Sin embargo, estas aproximaciones no son utilizables en los casos en los que las páginas y los datos contenidos en las mismas no siguen unas normas establecidas asimilables a las de un modelo Entidad-Relación. WebML se presenta como una aportación desde el punto de vista de los diseñadores Web y de la publicación estructurada y descriptiva de las páginas, sin abordar el tema de la integración de datos semiestructurados por parte

de sus usuarios o lectores.

Ya en el terreno del desarrollo de clientes Web que naveguen automáticamente, las herramientas más usadas para la automatización de la navegación de enlaces en el Web suelen ser programas completos (disponibles con múltiples opciones de ejecución) para descargar documentos del Web o realizar otro tipo de acciones sencillas desde la *Web superficial*. Muy usado en este sentido es Wget [20], pese a que sólo sirve para descargar documentos. Algunas herramientas como Curl [5] permiten un uso más avanzado al ofrecer al usuario la posibilidad de manejar también formularios además de enlaces, eso sí, solicitando al usuario datos de bajo nivel como el *query-string* que se debe enviar, en lugar de crearlo él a partir de una estructura de datos que represente a un formulario relleno. Estas últimas no ofrecen la posibilidad de recibir establecido el guión de una sesión completa HTTP más allá de una sola transacción, por lo que la secuencia de acciones de la tarea sólo puede ser especificada desde fuera de la herramienta, mediante llamadas a la misma, típicamente en un lenguaje de intérprete de comandos (shell) del propio sistema operativo, y siempre que no sea necesario el manejo de aspectos de bajo nivel, como son las cookies [95].

Algunas herramientas, como Veriweb [30], sí son capaces de seguir más de un enlace y formulario en una única ejecución, pero se limitan a realizar pruebas de ejecución sobre herramientas accesibles desde el Web, siguiendo sus enlaces y testeando que los caminos o tareas que cada una de esas aplicaciones permite realizar desde cada una de sus páginas no produzca errores en la aplicación del servidor ni provoque que al cliente le lleguen páginas de error (que son indeseables para muchos sitios comerciales porque menguan futuras visitas al *site*). En cualquier caso, no permiten al usuario la automatización de una tarea concreta, sino que recorren todos los caminos encontrables probando con distintos datos para rellenar los formularios, por lo que sus algoritmos de fuerza bruta no resultan adecuados para la automatización de tareas condiderables *de utilidad* para los usuarios.

Por otro lado, existen varias bibliotecas [39, 35, 23, 112, 118] sobre varios lenguajes de programación (Prolog, Perl, Java, C, ...) que sí permiten la ejecución de una secuencia preprogramada de transacciones Web, combinada a su vez con la extracción de datos desde documentos XML obtenidos por la red. En ocasiones, más que bibliotecas para algún lenguaje de programación, se proponen lenguajes propiamente de consulta para el Web [26, 51, 37, 93, 85]. Sin embargo, la gran mayoría de estas bibliotecas o lenguajes no permite el diálogo con servidores del Web *legado* por su falta de manejo de ciertas acciones implícitas que realizan los browsers sin que el usuario sea

consciente de ello, como las mencionadas en el apartado 2.1. También suelen aplicar técnicas de extracción de datos como son las expresiones regulares y la definición de analizadores léxicos que tratan a esas páginas como texto plano, sin estructura y que han sido utilizadas en trabajos como [47, 46, 57, 49, 100, 99, 98, 48, 45, 88]. Sin embargo, ninguna de estas soluciones propone una extracción de datos semiestructudados basada en XPath. A lo sumo, algunas de ellas, como [29, 80] definen sus propias primitivas de extracción de datos, pero en ningún caso basadas en el estándar del W3C.

Otras técnicas, como [81, 54, 80] resuelven bien la parametrización de acciones implícitas (ver apartado 2.1). Sin embargo, dejan no muy bien cubiertos desde el punto de vista de la mantenibilidad, aspectos tan importantes como lo es la extracción de datos del Web, una vez que la página que contiene finalmente los datos ha sido recuperada.

Desde el punto de vista de las aportaciones hechas por el W3C, varias técnicas, como XSLT [130] o DEL (Data Extraction Language) [133] han sido propuestas para la obtención en formato XML de datos extraíbles de otros documentos XML mediante reglas de extracción y transformación. Ambas definen etiquetas para crear fácilmente lenguajes de script que sean fácilmente interpretables. Sin embargo, carecen de mecanismos para indicar la forma de obtención de los documentos de los cuales deben extraerse los datos.

Por una parte, resulta interesante el enfoque que toman [54, 133, 130, 82] de definir lenguajes de programación sobre sintaxis XML en donde hay etiquetas capaces de representar variables, bucles, condiciones, y otros tipos de acciones. Este tipo de lenguajes de programación, definidos sobre esta sintaxis es fácilmente interpretable por varios programas y muy fácilmente analizable.

En el W3C, el tema de la extracción de datos se ha centrado últimamente en la definición del lenguaje de consulta XQuery, una extensión de XPath 2.0 (en realidad un superconjunto) que permite la consulta para la extracción de datos de cualquier documento XML, con funcionalidades avanzadas similares a las de otros lenguajes de consulta de bases de datos relacionales, como SQL [78]. Sin embargo, estas iniciativas, pese a que ofrecen muy buenas soluciones para ese problema (de hecho ésa es la razón por la que en esta tesis se haya partido de XPath 2.0 para crear un lenguaje de consulta y manipulación), no se enfrentan al tema de cómo obtener los documentos XML del Web ni tampoco acerca de cómo integrar los datos que aparezcan repartidos en distintos documentos.

De entre los últimos proyectos más avanzados para ser aplicados al Web *legado*, puede destacar [94, 111], en el que se aplican modernas técnicas de

extracción de datos semiestructurados a las páginas HTML, a páginas corregidas con Tidy [104]. En [94] se usa XSLT y en [111, 25] se considera al Web como una base de datos lo suficientemente estructurada como para poder usar lenguajes de consulta de bases de datos como XQL, un lenguaje de consulta predecesor de XQuery. Gracias a herramientas como Tidy, algunos usuarios pueden usar herramientas de evaluación de expresiones XPath como [21] sobre el Web legado. Sin embargo, tres importantes factores no han sido tenidos suficientemente en cuenta por estos trabajos.

- En primer lugar, no solventan adecuadamente las dificultades que afronta escoger XSL como lenguaje de manipulación de datos (se generan documentos de salida en lugar de manipular el árbol del documento y permitir el almacenamiento de sus partes en repositorios programables).

- En segundo lugar, no incorporan mecanismos de mejora a la robustez ante fallos, como los muy convenientes *Service Combinators* [40], siendo poco adecuados para la extracción de datos en páginas con estructuras de marcado cambiantes e irregulares.

- En tercer lugar, ninguno tampoco se ha enfrentado aún al uso del nuevo borrador de XPath 2.0 definido por el W3C para este fin, que incluye muchas e importantes mejoras respecto de la versión actual del estándar que salió a la luz en 1999.

Otros trabajos, como RoadRunner [55], pretenden aplicar algoritmos para generar automáticamente wrappers a partir de documentos de entrada. Pese a que ello supone un interesante enfoque capaz de minimizar los problemas de la generación manual y del mantenimiento de estos wrappers, la casi total falta de estructura esperable en los documentos del Web provocan que este tipo de soluciones sólo funcionen adecuadamente con ejemplos muy sencillos y muy regulares en su estructura, lo cual no siempre se puede es fácilmente encontrable en las aplicaciones del Web cuyo uso se desea automatizar.

Finalmente, uno de los más completos trabajos realizados hasta el momento en el campo de la integración de datos en el Web corresponde sin duda a [31], donde se tienen en cuenta muchos factores adecuadamente escogidos que facilitan el desarrollo de programas *envoltorio* por personas con pocas capacidades de programación. Una herramienta de fácil utilización permite dotar a estos programas de la conveniente robustez ante cambios en las páginas, minimizando los costes de su mantenimiento. En ese trabajo, dos lenguajes han sido desarrollados y soportados para dos propósitos bien distintos, pero complementarios entre sí: la obtención de documentos del Web

(mantenimiento del diálogo en una sesión HTTP con servidores Web) y la extracción de datos relevantes de cada uno de esos documentos. El primero de los lenguajes, NSEQL, es un lenguaje de alto nivel de abstracción en el que se indica la secuencia fija y preestablecida de acciones que deben activarse en un módulo navegador, una tras otra, para obtener del Web los documentos involucrados en una tarea. En dicha secuencia de acciones, se hace típicamente una referencia implícita al elemento o documento activo seleccionado por alguna acción anterior, de forma que, además de las acciones de seguimiento de enlaces, envío de formularios y rellenado del valor de sus campos, se contemplan otras más propias de la orientación a un browser como la *focalización* de documentos, formularios o elementos que quedan seleccionados como *activos* para que las siguientes acciones los puedan tomar como referencia. Otro lenguaje, llamado DEXTL se encarga de extraer los datos relevantes de cada una de esas páginas, de forma que el uso combinado de ambos lenguajes permite la automatización de prácticamente casi cualquier tarea en el Web.

Sin embargo, algunas características de estos lenguajes podrían ser mejorables.

- Para empezar, DEXTL es un lenguaje desarrollado para un generador de analizadores léxico-sintáctico propio, basado en expresiones regulares y en reglas gramaticales que indican la estructura esperada del texto para ser así correctamente reconocido y de esa forma poderle aplicar las correspondientes reglas de extracción de datos. Una opción quizá más interesante, podría haber sido la de aplicar tecnologías como XPath para poder seleccionar adecuadamente los nodos relevantes del documento. Ello supondría cierta pérdida de flexibilidad (XPath no podría ser aplicable a cualquier fichero de formato textual, como DEXTL sí lo permite), pero habría proporcionado quizá un mayor grado de robustez a las expresiones de extracción de datos en documentos HTML, puesto que una expresión XPath ocupa típicamente una única línea de código fácilmente entendible, en lugar de las varias que se necesitan para poder crear una expresión DEXTL equivalente.

- Por otra parte, el API de primitivas desarrolladas para NSEQL está demasiado particularizado para los eventos propios de un navegador. Pese a que proporcionan un avanzado esfuerzo por representar adecuadamente en el correcto nivel de abstracción las acciones que forman parte de la secuencia de eventos que un usuario genera con un browser durante un proceso de navegación, es decir de acciones básicas como las mencionadas en la tabla 2.5, este API podría haber sido un poco

más simplificado, ortogonalizado para hacerlo independiente de los elementos concretos de HTML y sus posibles eventos, y reorientado a la robustez. Dado que NSEQL no se basa en un modelo de datos como el de XPath, sino en los elementos y documentos activos que un navegador puede tener en cada momento, muchas de las funcionalidades del API se basan en la aplicación de algún evento a algún tipo concreto de elemento o atributo de HTML. Existe una función para seguir enlaces de *anchors*, que sin embargo no sirve para seguir enlaces de otro tipo de elementos, como *areas* (es decir, mapas de imágenes), que también tienen enlaces que a veces interesa poder seguir.

- La existencia de una referencia implícita al elemento o documento previamente seleccionado, y la ausencia de asociación explícita de nombres a las páginas previamente visitadas, impide, por ejemplo, que una misma página visitada con anterioridad a la página actual pueda ser *reutilizada* sin tener que abandonar el contexto de la actual. Por ejemplo, dentro de un bucle en el que ciertas páginas con formularios deban ser revisitadas para emprender con ellas el procesamiento de un nuevo conjunto de datos en una labor repetitiva, resulta necesario emular la acción *Back* del navegador, que puede implicar traerse una nueva versión de la página en lugar de reutilizar la que previamente había sido obtenida.

- Por otro lado, buena parte de las funciones del API necesitan recibir combinaciones de tres datos, típicamente un texto, un booleano y un número entero, indicando que se desea escoger el enésimo elemento de una lista de posibles que cumplan el tener al texto, bien contenido de forma exacta, bien contenido como subcadena, bien como texto, bien quizá en algún atributo concreto, todo ello dependiendo del argumento booleano y del nombre concreto de la función a la que se invoque. Dicho de otro modo, existen funciones para buscar elementos por texto o por algunos atributos, pero no para seleccionar elementos que cumplan una combinación arbitraria de cualquier nivel de complejidad con varios de ellos, abstrayéndose de nombres concretos de etiquetas o atributos particulares de HTML, como sí permiten los predicados de XPath.

- Por otro lado, el rellenado de formularios se plantea igualmente con un conjunto predefinido de funciones, cada una de las cuales permite realizar una acción concreta y predefinida sobre un tipo concreto de campo de formulario, cuando sería mucho más simple y flexible, aunque quizá también de más bajo nivel, permitir la modificación genérica de cualquiera de sus atributos. Por ejemplo, una de las funciones permite

rellenar un campo de texto de un formulario (modificando su atributo *value*, se entiende). Otras funciones permiten seleccionar la opción de una lista de opciones posibles. Por el contrario, no existen funciones para cambiar otros tipos de atributos influyentes en los campos de formularios, como *checked* o *disabled*.

- En prácticamente casi todas las funciones, es imprescindible la indicación de un número entero que indique la posición del elemento que se desea direccionar dentro de una lista de otros que cumplan una serie de características. Así, por ejemplo, se puede seguir el séptimo enlace de una página, por ejemplo. Sin embargo, el criterio de la posición de un elemento dentro de su página es altamente sensible a cambios en las páginas HTML, pues en el momento en el que la página del servidor añada una nueva opción a la lista de opciones, las posiciones de éstas se ve irremediablemente trastocada, provocando que en la labor de mantenimiento de estos programas se deba proceder a la actualización de estos valores numéricos. Una selección basada en contenidos independientes de posiciones es más robusta o, en cualquier caso, más fácilmente mantenible.

En cualquier caso, el API está más orientado a los eventos que espera recibir un browser gráfico como lo es *Microsoft Internet Explorer*, que al propio diálogo HTTP que espera el servidor al que se accede remotamente, puesto que no se dialoga directamente con el servidor sino que de delega esta labor a este browser concreto. Sin embargo, el uso de un browser gráfico en la plataforma de ejecución le aporta una indudable ventaja: y es que este sistema tiene incorporada la ejecución de rutinas JavaScript, algo que no tienen todas las plataformas de navegación automatizada.

3.4.1. Uso de APIs estándares

Por otro lado, programar con APIs estándares tampoco es una garantía de éxito si ello implica la utilización inadecuada de la tecnología. Por ejemplo, en [115] se explica lo dificultoso que puede ser realizar un programa Java que extraiga datos de un documento XML tan sencillo como el de la figura 3.3.

Un programa Java que use DOM y quiera extraer las direcciones (elementos `address`) de aquellas personas que coinciden con su argumento puede ser el que aparece en la figura 3.4.

Teniendo en cuenta que existe una expresión XPath `//address[child::addressee[text() = 'Jim Smith']]` capaz de

```
<addressbook>
   <address>
      <addressee>John Smith</addressee>
      <streetaddress>250 18th Ave SE</streetaddress>
      <city>Rochester</city>
      <state>MN</state>
      <postalCode>55902</postalCode>
   </address>
   <address>
      <addressee>Bill Morris</addressee>
      <streetaddress>1234 Center Lane NW</streetaddress>
      <city>St. Paul</city>
      <state>MN</state>
      <postalCode>55123</postalCode>
</address>
</addressbook>
```

Figura 3.3: Ejemplo de documento XML sencillo

```
public Node findAddress(String name, Document source) {
   Element root = source.getDocumentElement();
   NodeList nl = root.getChildNodes();

   // iterate over all address nodes and find the one that has the correct addressee
   for (int i=0;i<nl.getLength(); i++) {
      Node n = nl.item(i);
      if ((n.getNodeType() == Node.ELEMENT_NODE) &&
          (((Element)n).getTagName().equals("address"))) {
         // we have an address node, now we need to find the
         // 'addressee' child
         Node addressee = ((Element)n).getElementsByTagName("addressee").item(0);

         // there is the addressee, now get the text node and compare
         Node child = addressee.getChildNodes().item(0);
         do {
            if ((child.getNodeType()==Node.TEXT_NODE) &&
                (((Text)child).getData().equals(name))) {
               return n;
            }
            child = child.getNextSibling();
                  } while (child != null);
      }
   }
   return null;
}
```

Figura 3.4: Programa Java que extrae datos de documento XML con DOM

obtener la dirección completa de Jim Smith y que existen bibliotecas Java capaces de evaluar expresion XPath, una buena opción podría ser reescribir en programa de la figura 3.4 por el de la figura 3.5.

```
public Node findAddress(String name, Document source) throws Exception {

   XPathHelper xpathHelper = new XPathHelper();
   NodeList nl = xpathHelper.processXPath(
        "//address[child::addressee[text() = '"+name+"']]",
        source.getDocumentElement());
   return nl.item(0);
}
```

Figura 3.5: Programa Java que extrae datos de documento XML con XPath

Como puede verse, el programa resulta mucho más sencillo por hacer uso de XPath. XPath, del que se detallan ventajas e inconvenientes en el apartado 4.2, suele usarse normalmente embebido en un lenguaje anfitrión que normalmente es XSLT. Una hoja XSLT que realiza lo mismo que el programa Java anterior es la que aparece en la figura 3.6. Como puede verse, una hoja XSLT puede ser también muy verbosa y demasiado compleja incluso para tareas sencillas y saber escoger un estándar adecuado es primordial para mantener bajo el coste de mantenimiento de los programás.

Por otro lado, XML-Binding [34] resulta una tecnología novedosa para la manipulación de documentos XML con un interfaz más simple que el proporcionado por DOM. La idea principal de XML-Binding consiste en generar clases compilables en un lenguaje de programación, típicamente Java, automáticamente a partir de su descripción en XML Schema [134]. De esta forma, los documentos que validen con ese XML Schema son fácilmente procesables por objetos de la clase correspondiente. Ello es útil en entornos donde DOM ofrece un acceso costoso a los programadores que quieren tener una visión lógica de los datos en lugar de una visión en árbol del documento. Su utilidad es aplicable especialmente en entornos donde los lenguajes XML Schema son desconocidos a priori o pueden sufrir modificaciones, de forma que esos cambios quedan ocultados por la aplicación generadora de clases. Sin embargo, este tipo de trabajos no es directamente aplicable al Web legado basado en HTML.

```
<?xml version='1.0'?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
   <xsl:output method="xml"/>

<xsl:variable name="doc-file">http://mymachine.com/changed.xml</xsl:variable>

<!-- copy everything that has no other pattern defined -->
<xsl:template match="* | @*">
   <xsl:copy><xsl:copy-of select="@*"/><xsl:apply-templates/></xsl:copy>
</xsl:template>

<!-- check for every <address> element if an updated one exists -->
<xsl:template match="//address">
   <xsl:param name="addresseeName">
      <xsl:value-of select="addressee"/>
   </xsl:param>

   <xsl:choose>
      <xsl:when test="document($doc-file)//addressee[text()=$addresseeName]">
         <xsl:copy-of select="document($doc-file)//address[child::addressee[text()=$addresseeName]]"/>
      </xsl:when>
      <xsl:otherwise>
         <xsl:apply-templates/>
      </xsl:otherwise>
   </xsl:choose>
</xsl:template>

</xsl:stylesheet>
```

Figura 3.6: Hoja XSLT que extrae datos de documento XML

3.5. Conclusiones del estado de la cuestión

La tabla 3.1 refleja un resumen de las tecnologías mencionadas en este capítulo, comparando unas con otras respecto a varios criterios comunes. La primera columna corresponde a las soluciones *ad hoc*, como [26, 51, 37, 93, 85], basadas en expresiones regulares y en plataformas creadas específicamente para resolver problemas concretos. La segunda columna se corresponde con la opción de usar un parser genérico tipo DOM al estilo del empleado en la figura 3.4, utilizable en una biblioteca utilizable desde un lenguaje de programación. La tercera columna, muy similar a la segunda, refleja la opción de usar JavaScript circunscribiéndose a un browser, en tanto que también usa DOM, salvo que con otra sintaxis distinta a la de Java. La cuarta opción se corresponde con plataformas al estilo de Xalan [62], es decir, parsers tipo DOM pero capaces de evaluar expresiones XPath, lo cual simplifica mucho la labor de programación, tal y queda reflejado en la figura 3.5. La quinta columna, se corresponde con la opción de usar WebL [80] y la sexta se corresponde con los resultados de esta tesis a efectos comparativos.

	Ad hoc	Parser	JavaScript	Xalan	WebL	XPlore
Estándar	Exp. Reg.	DOM	DOM	XPath+DOM	No	MSC+XPath
Extracción	Bajo	Bajo	Alto	Alto	Alto	Alto
Navegación	Alto	Alto	Browser	Alto	Alto	Alto
Plataforma	Mala	Mala	Browser	Mala	Media	Buena
Simplicidad	No	No	Sí	Sí	Sí	Sí
Rutinas	Sí	Sí	Sí	Sí	Sí	Sí
Robustez	No	No	No	No	Sí	Sí
Localización	Sí	No	Sí	Sí	Sí	Sí
Legibilidad	Mala	Buena	Buena	Buena	Buena	Buena
HTML legado	Ad hoc	Tidy	Sí	Tidy	Sí	Tidy
XML	Ad hoc	Sí	No	Sí	Sí	Sí

Cuadro 3.1: Resumen de las tecnologías utilizables

Como puede apreciarse, la extracción de datos presenta un bajo nivel de abstracción cuando se emplean expresiones regulares o un parser tipo DOM sobre lenguajes de programación convencionales (salvo la contada excepción de JavaScript). A pesar de que todas las opciones presentan un nivel de abstracción adecuado para representar las acciones navegacionales del protocolo

HTTP, pocas de ellas ofrecen un buen soporte a los aspectos de más bajo nivel de dicho protocolo (acciones implícitas mencionadas en el apartado 2.1), por lo que no son utilizables para el establecimiento de sesiones con servidores Web. Por otro lado, aunque de uno u otro modo, sea posible la utilización de rutinas de usuario para las acciones más complejas, no siempre es posible definir de forma simple el comportamiento en las acciones más sencillas. Por ejemplo, las soluciones *ad hoc* suelen basar casi todo su funcionamiento en rutinas definibles por el usuario. En cuanto a la robustez ante fallos en la conexión, pocas opciones ofrecen mecanismos de recuperación ante fallos. Por otro lado, la facilidad de localización de las zonas de código afectadas por un posible cambio en la estructura de las páginas puede ser complicada si se usa un parser tipo DOM, (debido al alto número de líneas de código involucradas en cada acción) teniendo en cuenta, no obstante, que su legibilidad suele ser, pese a esa verbosidad, aceptable. Algo completamente distinto suele ocurrir con las expresiones regulares, que suelen concentrar en una zona de código muy concreta el lugar afectado por un cambio en las páginas, aunque sin embargo proporcionan una mala legibilidad de las mismas debido a su bajo nivel de abstracción. Finalmente, unas u otras opciones son aplicables a cualquier formato, tanto HTML del Web legado como XML, pudiendo requerir para ello herramientas externas como [104].

Como resumen, las soluciones actuales mencionadas en este capítulo adolecen de uno o varios de los siguientes problemas:

No basadas en estándares

La mayoría de los trabajos relacionados suelen afrontar el problema de la integración de datos semiestructurados desarrollando lenguajes *ad hoc* para propósitos específicos, cada uno de ellos adaptado a sus propios propósitos específicos. La gran mayoría de ellos suelen estar basados en analizadores léxicos basados en expresiones regulares, aplicadas sobre los documentos como si fueran ficheros de texto plano, ignorando la estructura lógica de marcas del documento, al contrario que como ocurre con la utilización de otros estándares conocidos. Muchos de estos proyectos, o bien han sido abandonados, o bien han tenido poco éxito o repercusión en la comunidad, o bien están siendo explotados bajo un elevado coste de mantenimiento. Por el contrario, una solución basada en estándares resulta más fácilmente mantenible debido a una mayor legibilidad y un mayor nivel de difusión.

Bajo nivel de abstracción

Los trabajos que sí suelen optar por una solución basada en estándares suelen muchas veces tropezar con una inadecuada elección de la tecnología que va a utilizar. Así pues, de los proyectos relacionados que sí han optado desde un principio por la utilización de estándares XML en sus desarrollos, la gran mayoría ha escogido usar tecnologías como SAX [86], DOM [131] (o similares) [77, 72] o XSLT [130]. Cada una de estas tecnologías presenta diferencias importantes en cuanto a la potencia, eficiencia y nivel de abstracción. Por ejemplo, SAX tiene un nivel de abstracción muy bajo pero eficiente (útil para tareas sencillas sobre datos voluminosos), mientras que el de XSLT es más elevado. Por otro lado, DOM resulta ser la técnica más potente de las tres, pero también la que requiere más conocimientos y experiencia, y por lo tanto, dedicación y recursos. XSLT, diseñado como un lenguaje de especificación de transformaciones y, pese a su gran aceptación, resulta demasiado sofisticado y verboso cuando las reglas de transformación que maneja adquieren un poco de complejidad (muchos bucles, variables tipadas, condicionales y expresiones calculando valores temporales para ciertos cálculos), en cuyo caso un lenguaje de programación imperativo puede ser mejor opción. En cualquier caso, estas tres tecnologías adolecen de un mismo problema, y es que todas ellas comparten el hecho de requerir varias líneas de código para una operación que en principio podría ser considerada como sencilla.

Baja escalabilidad

Los sistemas desarrollados hasta el momento no intentan minimizar la complejidad del código particularizado de acceso a los servidores Web (código envoltorio) [83], por lo que los sistemas desarrollables con estas técnicas resultan ser muy poco escalables, es decir, muy costosos de desarrollar y mantener en cuanto el número de servidores a los que se accede crece.

Baja adaptabilidad ante cambios

Los sistemas desarrollados hasta el momento no intentan favorecer, en las aplicaciones desarrollables con ellos, la adaptabilidad ante cambios en las páginas del Web. Por esa razón, es común, que estas aplicaciones desarrolladas con estas técnicas acaben dejando de funcionar por la aparición de pequeños y frecuentes cambios, muchos de ellos incluso imperceptibles para los usuarios si se accede con un browser.

Limitaciones en la capacidad de construcción

Buen número de funcionalidades, principalmente acciones implícitas necesarias para mantener correctamente la sesión con servidores Web, no están completamente contempladas por la gran mayoría de plataformas de desarrollo existentes, por lo que muchos programas desarrollados acaban por no poderse utilizar de forma efectiva en numerosos servidores del Web *legado*.

Alto coste de mantenimiento

Los productos tienen una vida corta, normalmente hasta que el Web al que acceden cambie algo que rompa con las suposiciones establecidas implícitamente por el programador del código envoltorio, por lo que necesitan ser revisados cada poco tiempo, siendo además costoso localizar y corregir en el código del programa el cambio que ha provocado que la aplicación haya dejado de funcionar. Como el Web es un ente cambiante que evoluciona dinámicamente, resulta bastante imprevisible el momento en el que se va a detectar un fallo o el impacto que un cambio va a suponer en un algoritmo de navegación.

Navegación superficial

El denominado *deep Web* [110, 105], esto es, el Web obtenible del volcado de bases de datos, está aún por explotar convenientemente, ya que los sistemas de navegación existentes apenas permiten seguir enlaces fácilmente obtenibles, como los que tienen una URL estática o aquellos para los que no hace falta rellenar apenas formularios.

3.6. Limitaciones de las tecnologías actuales

Buena parte de los proyectos que abordan el tema de la automatización de la navegación en el Web, al igual que este trabajo, reconociendo igualmente que quizás los beneficios de iniciativas como las de XML o la mejora de la accesibilidad tardarán en poderse apreciar, plantean sus propias soluciones directamente sobre las páginas HTML del Web *legado*. Buena parte de esos trabajos plantean sus principios considerando a las páginas Web como ficheros de texto plano, de forma que a las páginas Web obtenidas se les extraen los datos aplicando patrones basados en expresiones regulares ba-

sadas en el código HTML que rodea a cada dato que se desea extraer. Si bien la aplicación de expresiones regulares resulta ser una técnica lo suficientemente potente como para poder ser aplicada también a otros tipos de formatos textuales, lo cierto es que las expresiones regulares resultan ser una técnica de bajo nivel de abstracción, donde, toda vez que pequeños aspectos sintácticos tales como la aparición inhabitual de múltiples espacios en blanco o fines de línea, la ambivalencia de comillas dobles y simples, la indistinción de mayúsculas y minúsculas o el orden de aparición de los atributos de una etiqueta hayan podido ser superados, la legibilidad de esas expresiones se complica notablemente. Por otro lado, mediante el uso de esas expresiones regulares, la estructura de árbol de la página HTML no es considerada. Este hecho puede no suponer un problema si la estructura de la página HTML y la consulta que se le desea efectuar son lo suficientemente simples. No obstante, es deseable a veces poder aplicar patrones de extracción de datos a determinados fragmentos específicos de la página, es decir, a cierto número de subárboles que cumplan determinadas propiedades (porque se sabe que sólo se desea buscar en determinadas zonas relevantes de la página), en lugar de a todo el documento. Para estos casos, las expresiones regulares resultan ser bastante limitadas.

Los últimos proyectos que abordan estos problemas han descubierto en la familia de especificaciones XML una tecnología adecuada con el que poder enfrentarse al problema anterior de una forma mucho más elegante y robusta. Para poder aplicar este tipo de soluciones se necesita, no obstante, de un software capaz de *reparar* los errores sintácticos habituales en una gran cantidad de páginas Web. Dichos errores sintácticos, tales como la falta de cierre de etiquetas, el incorrecto anidamiento de las mismas o la ausencia de entrecomillado en los atributos deben ser corregidos en las páginas según éstas van siendo obtenidas de los servidores. Los documentos HTML así obtenidos quedan transformados en sus equivalentes documentos XHTML [117] antes de poder aplicarles cualquier técnica XML. Si bien existen numerosos programas [58, 146] capaces de obtener una versión de la página donde se cumplan los principios de *buena formación* de XML, esto es, conforme a la sintaxis básica de XML, la herramienta Tidy [104] suele ser la más habitualmente usada.

Los pocos trabajos que emplean el enfoque de reparación sobre la marcha de la sintaxis de los documentos (con herramientas como [104, 58, 146]), suelen aplicar a las páginas Web obtenidas, una vez ya corregidas, bien parsers tipo DOM [131] sobre los que aplicar directamente código programado en lenguajes convencionales de programación (típicamente Java), o bien hojas de estilo XSLT [130] para extraer de las páginas la información que les interesa.

Cuando las hojas de estilo XSLT no ofrecen toda la funcionalidad deseable para el problema que desean afrontar, estos sistemas suelen permitir la aplicación de funciones de usuario escritas en algún lenguaje de programación convencional. Si bien la estructura declarativa de las hojas XSLT permite definir un gran número de tratamientos aplicables a los documentos XHTML de entrada, muchas veces el tratamiento aplicable a ciertas páginas requiere algo más que la simple extracción de datos en documentos de salida. Esa extracción por sí sola puede ser suficiente en tareas simples de recuperación de información, pero es insuficiente en aquellas tareas que deban manejar estructuras de datos que recopilen datos del usuario, como son los formularios. En ocasiones resulta necesaria la manipulación mediante inserción y borrado de nodos y/o atributos o la manipulación repetida de esos atributos en un mismo documento. Los formularios Web son muchas veces un claro ejemplo que demuestra, para la tarea de ser rellenados múltiples veces, la necesidad de una manipulación sencilla y eficiente de pequeñas partes de un documento, donde la referencia de un único árbol accesible representativo del documento sea constantemente visible sin la necesidad de estar creando y destruyendo subárboles mediante la aplicación de sucesivas hojas XSLT.

Si bien XSLT es una solución entendible por numerosos programadores, el actual borrador de XPath 2.0 [141] está suponiendo serios replanteamientos acerca de lo que se espera que serán las futuras versiones de XSLT. Gran parte de la expresividad de XSLT 1.0 está sustentada en la capacidad de la robusta extracción de datos de XPath 1.0. XPath 2.0, gracias a la incorporación de nuevos operadores que no aparecían en la anterior versión y su integración dentro del marco de XQuery [142] como lenguaje de consulta, está constituyéndose como un potente mecanismo de direccionamiento de datos en documentos XML, suficiente por sí mismo como para no necesitar la funcionalidad extra de XSLT a la hora de realizar integradores de datos semiestructurados. De hecho, XPath puede considerarse, a pesar de ciertas limitaciones de las que adolece, como un mecanismo de extracción y direccionamiento de mayor nivel de abstracción que el mero uso de expresiones regulares. Una de las contribuciones de este trabajo, de hecho, consiste, aparte de una plataforma de implementación que evalúa expresiones del borrador de XPath 2.0, en un conjunto de extensiones propuestas para extender XPath 2.0 con el fin de convertirlo en un lenguaje completamente funcional y eficaz para esta labor y otras, como la manipulación de documentos.

Por otro lado, buena parte de los trabajos relacionados mencionados en la bibliografía han hecho uso de clientes simples del protocolo HTTP, la mayoría de ellos disponibles fácilmente desde diversas bibliotecas o programas y en diversos lenguajes de programación. Si bien la mayoría de ellos son ca-

paces de proporcionar una funcionalidad básica aceptable, lo cierto es que muchas de las técnicas actualmente empleadas por algunos servidores Web para mantener el concepto de sesión con los clientes que a ellos se conectan, resultan no ser tenidas en cuenta por estas soluciones. Si bien las cookies suelen tener un comportamiento genérico normalizado y es posible su fácil implementación en los clientes HTTP al estar bien recogidas y localizadas entre las cabeceras de este protocolo, otras técnicas, como ciertos identificadores de sesión, que se pueden encontrar ocultos entre los parámetros de las páginas Web, pueden fácilmente hacer malfuncionar el comportamiento de un cliente HTTP que no los contemple debidamente. Dicho de otro modo, los actuales clientes HTTP utilizables para los procesos de navegación automática tienen un bajo nivel de soporte para el conjunto de acciones implícitas que otras herramientas como los browsers gráficos sí realizan bien. Cabe destacar que aquellos enlaces que no figuran explícitos según las habituales normas de HTML, sino que funcionan como resultado de la computación de alguna rutina de JavaScript o similares, resultan especialmente problemáticos en tanto en cuanto prácticamente no existe ningún tipo de soporte para estos lenguaje en muchos de estos clientes HTTP. Ello provoca que para muchos usuarios la única forma de acceder a determinados contenidos sea a través de un browser.

Existen numerosos proyectos que han abordado la temática de la navegación automática y de la integración de datos del Web a lo largo de los últimos años. Todos ellos, incluyendo el presente, se centran en la creación de *wrappers* o código de programa especializado en el tratamiento particularizado de cada fuente de datos. Afrontar el problema desarrollando un único programa capaz de navegar en cualquier sitio Web, capaz a su vez de solventar todas las heterogeneidades de cada servidor, resulta algo impracticable en tanto en cuanto se necesita para ello introducir semántica necesaria para poder autodetectar la información necesaria para la navegación, manejando aspectos como sinónimos y reconociendo al momento los enlaces que se deben seguir y los formularios (campos incluidos) que se deben rellenar, lo cual resulta demasiado complicado incluso para tareas sencillas. El Web Semántico es una buena alternativa que intenta basar este reconocimiento semántico en el uso de metadatos que permitan ser utilizados para la deducción acerca de cómo efectuar las acciones básicas explícitas de la navegación (ver apartado 2.2). Sin embargo, el Web Semántico aún tiene importantes flecos que deben resolverse, y, en cualquier caso, no parece estar enfocado al Web *legado*, sino a un nuevo Web creado a partir de esos metadatos.